December 2020

Reinforcement Learning for Mobile Robot Collision Avoidance in Navigation Tasks

Zilong Jiao
Syracuse University

Abstract

Collision avoidance is fundamental for mobile robot navigation. In general, its solutions include: *map-based* and *mapless approaches.* In the map-based approach, robots pre-plan collision-free paths based on an environment map and follow their paths during navigation. On the other hand, the mapless approach requires robots to avoid collisions without referencing to an environment map. This thesis first studies the map-based approach for multiple robots to collectively build environment maps. In this study, a robot following a pre-planned path may encounter unexpected obstacles, such as other moving robots and obstacles inaccurately presented on an environment map. This motivates us to study mapless collision avoidance in the second part of the thesis. Mapless collision avoidance requires a robot to infer an optimal action based on sensor data and operate in real time. Inferring an optimal action in a timely manner is computationally expensive, particularly when a robot has limited on-board computing resources. To avoid the expensive online action inferring, this thesis presents a reinforcement learning approach which learns policies for mapless collision avoidance under real-world settings. We first propose a Real-Time Actor-Critic Architecture (RTAC) to support asynchronous reinforcement learning under real-time constraint. Based on RTAC, we propose asynchronous reinforcement learning methods for mapless collision avoidance of various numbers of robots under different environment configurations. Through extensive experiments, we demonstrate that RTAC serves as a solid foundation to support multi-task and multi-agent learning for mapless collision avoidance under asynchronous settings.

Reinforcement Learning for Mobile Robot Collision Avoidance in Navigation Tasks

by

Zilong Jiao

B.E. Nanjing University of Posts and Telecommunications, 2011

B.F.A. New York Institute of Technology, 2011

M.S. Syracuse University, 2014

Dissertation

Submitted in partial fulfillment of the requirements for the Degree of

Doctor of Philosophy in Computer and Information Science and Engineering.

Syracuse University

December 2020

Acknowledgements

I would like to thank my advisor, Prof. Jae C Oh, for his patience and guidance throughout my PhD study. Besides my advisor, I would like to thank the rest of my thesis committee: Prof. Chilukuri K Mohan, Prof. Garrett E Katz, Prof. Qinru Qiu, Prof. Vir Virander Phoha, and Prof. Young Bai Moon for their insightful comments and enlightenment. Last but not the least, I would like to thank my parents for their support throughout this long journey.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The applications of mobile robots, such as autonomous cars and delivery robots, are becoming increasingly available and making significant social impact. The deployment of mobile robots in practice is safety-critical and requires robots to navigate in environments with high-level uncertainties. As a collision of a mobile robot can be catastrophic, it is crucial to ensure the safety of the robot and avoid damage to humans and properties in its surrounding environment.

Collision avoidance is an important problem and has been actively studied in robotics research. In general, a robot can avoid collisions through map-based and mapless approaches. This thesis considers a map as a grid representation of a physical environment that describes the characteristics of obstacles. In the map-based approach, robots plan collision-free paths first and follow the pre-planned paths during navigation. On the other hand, the mapless approach allows a robot to plan its motion based on local sensor data and take action in real time.

In a navigation task, a robot needs both map-based and mapless approaches to avoid collisions. This thesis first studies map-based collision avoidance in navigation tasks involving multiple robots. In this part of study, robots need to collectively build an environment map, so that they can plan collision-free paths for safe navigation. To

build an environment map, the robots adopt the *frontier-based exploration strategy* [99, 11]. Specifically, the strategy allows each robot to plan collision-free paths to different way-points (i.e., frontiers) and progressively build a map for the unknown regions of an environment. In this study of map-based collision avoidance, robots following different pre-planned paths may encounter each other during navigation. In this case, each robot considers the other robots as dynamic obstacles. However, the environment map built by the robots does not incorporate such dynamics. Besides, in practice, such an environment map is not perfectly accurate and can mistakenly present the locations of obstacles [90]. Therefore, during navigation robots must be able to avoid collisions with encountered obstacles in real time, and it is important to ensure safe navigation through mapless collision avoidance methods.

The rest of the thesis focuses on mapless approach collision avoidance, where robots operate in a continuous environment and take actions in real time. The conventional robotics solutions formulate mapless collision avoidance as an optimization problem, where a robot infers an action by searching its action space at each time step [24, 46, 54, 94, 95]. Such online action inferring can be computationally expensive, particularly when a robot has limited computing resources. This thesis presents a reinforcement learning approach which avoids the expensive online action inferring and learns a policy for a robot to map its sensor data to optimal actions for mapless collision avoidance. Such a policy is termed as a deterministic policy in reinforcement learning literature, as it allows a robot to deterministically select an action at each time step based on sensor observation.

Historically, reinforcement learning held its promises on learning optimal policies for small scale problems, but it was unclear about its applicability to mapless collision avoidance under real-world settings. Developing control solutions for real-world environments is challenging and must deal with the following issues, including par-

tial observability [84], high-dimensional sensor data [51], real-time control delay [19], multi-task environments [36], and multi-agent interaction [37].

Recently, deep reinforcement learning achieved impressive results in game-playing [57, 78, 9] and demonstrated its potential of contributing to mobile robot applications [49, 83, 15, 16]. However, none of these existing solutions consider all of the issues mentioned above. Applying reinforcement learning to real-world environments typically involves learning a policy in simulation and deploying the learned policy in the second phase [91]. Unlike the previous methods, this thesis addresses all the issues mentioned above and learns deterministic policies for mapless collision avoidance under real-world settings.

In a real-world environment, the data perceived by a robot through on-board sensors is usually high-dimensional. As on-board sensors have limited sensing ranges, the robot can only observe its surrounding areas, which makes its environment partially observable. Recent reinforcement learning literature shows that using a policy represented as a recurrent neural network can enable a robot to better estimate its underlying environment state [27]. To overcome partial observability and high-dimensional sensor data, we represent a robot's policy as a deep recurrent neural network which allows the robot to determine an action based on a temporal sequence of sensor observation.

A robot navigating in the physical world may need to deal with different scenarios and cooperate with other robots. Specifically, we define a scenario as an environment (or a region of an environment) which has a known obstacle distribution. This makes the physical world be a multi-task multi-agent environment [36, 37]. Here, a task refers to collision avoidance in a particular scenario. Learning an optimal policy for mapless collision avoidance requires progressively solving collision avoidance for 1) a single robot navigating in a particular scenario, 2) a single robot navigating in multiple scenarios, 3) multiple robots navigating in a single scenario, 4) and multiple

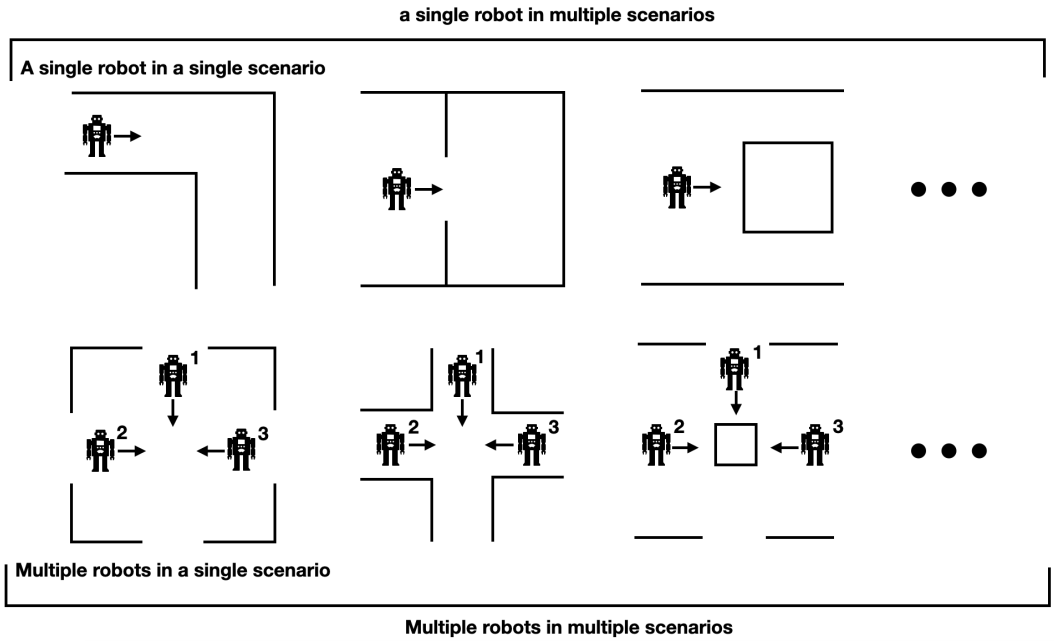


Figure 1.1: The examples of the mapless collision avoidance under different settings. In the examples, a scenario is an environment (or a region of an environment) with a specific obstacle distribution. For a multi-scenario setting, a robot or robots must avoid collisions in multiple different scenarios during navigation.

robots navigating under multiple scenarios. Figure 1.1 illustrates examples of those collision avoidance problems. This thesis presents reinforcement learning methods which address the above collision avoidance problems, from the simplest single-agent single-task setting to the most difficult multi-agent multi-task setting.

For mapless collision avoidance, a robot operates in real-time and must determine an action within a predefined time limit. It is unavoidable for the robot to have a varying time delay before actuating actions. In the context of reinforcement learning, we defined the time delay before a robot actuates an action at each time step as control delay. Through our empirical experiments and theoretical analysis, we found that control delay with high variance can greatly destabilize reinforcement learning and can even make an environment Non-Markovian. To reduce the variance of control delay, we proposed a **R**eal-**T**ime **A**ctor-**C**ritic architecture (RTAC) which allowed a robot to conduct policy learning and environment interaction with two separate

threads. Utilizing the recurrent neural network policy, RTAC enabled asynchronous reinforcement learning to learn a policy under real-time constraint. The learned policy allowed a single robot to avoid collisions in a single scenario. Based on RTAC, we proposed the method, Asyn^{d}-DDPG, to learn multi-task policies for a single robot to avoid collisions in multiple scenarios. Asyn^{d}-DDPG can asynchronously learn a policy with robots under different scenarios. By extending the method, we further proposed Rec-MADDPG, which can conduct the asynchronous learning based on the joint states and actions of robots. Rec-MADDPG can effectively handle multi-agent interaction and learn multi-agent policies for multiple robots to avoid collisions in multiple scenarios.

We evaluate the proposed work in simulation where robots are configured closely based on physical mobile robots (i.e., ROSbot and AR.Drone). In experiments, the simulated robots conduct navigation in physics-enabled maze-like environments. To support real-time operation, a reinforcement learning method is executed in a separate process that runs in parallel with robot simulation. This allows a simulated robot to update its motion while determining an action continuously. We evaluate RTAC in terms of its capability of stabilizing reinforcement learning and the scalability of asynchronous reinforcement learning. In the multi-task learning with Asyn^{d}-DDPG, we consider tasks as different mapless collision avoidance behavior environments with predefined obstacle distributions (i.e., scenarios). An effective multi-task policy must enable a robot to have a balanced performance on learned tasks and safely navigate environments consisting of the predefined obstacle distributions. In the evaluation of Rec-MADDPG, robots in the same environment need to navigate to their own goals. With effective policies, the robots shall be able to plan coordinated actions during navigation and reach their goals without collisions.

Through experiments, we demonstrated that RTAC can reduce variance of control delay and well support asynchronous reinforcement learning under real-time con-

straint. Besides, Asyn^{d}-DDPG can consistently synthesize the knowledge learned in different tasks into a balanced policy. The policy allowed a robot to navigate without collisions, even in the unseen environments during training. By projecting the joint observation of robots into low-dimensional embedding, Rec-MADDPG was able to effectively learn individual policies for the robots in a cooperative environment. With those policies, the robots can plan coordinated actions to avoid collisions and navigate to their goals in different environments. In extensive experiments, RTAC served as a solid foundation and allowed both Asyn^{d}-DDPG and Rec-MADDPG to learn effective multi-task and multi-agent policies in real-world settings.

We structured the rest of the thesis as follows. Chapter 2 presents the background related to collision avoidance and reinforcement learning. Chapter 3 introduces the motivating work based on map-based collision avoidance. Chapter 4 provides the formal definitions of the problem of mapless collision avoidance. Chapter 5 presents the architecture for asynchronous reinforcement learning under real-time constraints. Based on the architecture, Chapters 6 and 7 present the methods for learning multi-task and multi-agent policies for mapless collision avoidance in real-world settings. Finally, chapter 8 concludes the thesis.

Chapter 2

Background

2.1 Collision Avoidance

A collision of a mobile robot can be catastrophic, and it can cause severe damage to humans and properties in the robot's surrounding environment. Collision avoidance is fundamental for robot navigation, and its solutions can be broadly categorized as map-based and mapless approaches.

In the map-based approach, a robot has a map of an environment and can plan a collision-free path to its goal based on the map. Planning a path on an environment map has been extensively studied, and conventional techniques are based on A* and D*. During navigation, a robot follows a pre-planned plan to its goal. The map-based approach assumes that a robot can localize itself through on-board sensors while navigating in an environment. For localization, Kalman filters and particle filters are proven techniques for a robot to practically estimate poses in a wide range of applications. In this thesis, the map-based approach assumes that a robot can localize itself based on an available environment map and follow a pre-planned path during navigation. In practice, an environment map usually contains high-level uncertainties which are caused by dynamic obstacles and noisy sensors of robots. In this case, a

robot following a pre-planned path may encounter obstacles that are inaccurately represented on the environment map. Therefore, it is important for a robot to use the mapless approach to avoid collisions with unforeseen obstacles.

In the mapless approach, a robot operates in real time and avoids collisions based on on-board sensor data. Conventional robotic methods formulate the mapless collision avoidance as a real-time optimization problem. In those methods, a robot needs to search for an optimal action in its action space (or configuration space) within a pre-defined time limit. Those methods are commonly based on the techniques of artificial potential field [39], dynamic window [24] and velocity obstacle [95]. Such online action inferring can be computationally expensive for a robot with limited on-board computing resources. It is attractive to avoid the expensive action inferring by deterministically map robot sensor observation to optimal actions. This is where the advantages of reinforcement learning arises. Through reinforcement learning, a robot can learn a policy representing the deterministic mapping through offline trails and errors. However, applying reinforcement learning to mapless collision avoidance is challenging and often requires a robot to deal with the issues, including high-dimensional sensor data, partial observability, real-time operation, multi-task environments and multi-agent interactions.

In this chapter, we reviews the conventional map-based and mapless collision avoidance methods, according to different numbers of robots and characteristics of obstacles. In general, the characteristics of obstacles can be static and dynamic. Here, static obstacles are the obstacles with unchangeable shape and size. They persist in an environment at a fixed location all the time. On the other hand, dynamic obstacles are the ones that can move and can have variable shapes and sizes. Besides, they may not persist in an environment all the time. With the presented review, this chapter intends to highlight the need of having reinforcement learning as an

alternative approach for collision avoidance, which can avoid expensive online action inferring and learn effective behavior under real-world settings.

2.1.1 Single-Robot Collision Avoidance

In this section, we start reviewing the single-robot collision avoidance methods for environment that only contains static obstacles. Then, we will review the single-robot collision avoidance methods which address the environments with dynamic obstacles.

An environment can consist of arbitrary number of static obstacles. For navigation in the environment. a robot must have a goal to reach. With an environment map, a single robot can find an optimal collision-free path to its goal. Conventionally, the robot can plan a collision-free path using A*, D*, Rapidly Exploring Random Tree (RRT) [46] or Genetic Algorithm (GA). A* and D* algorithm is the standard path-finding algorithm based on Dijkstra's algorithm. RRT let a robot randomly sample the state space (the state of the robot consisting of the position and velocity), and iteratively build up a tree of sampled states to find the collision-free path to its goal. With GA, a robot represents its path as a sequence of way-points, which is considered as an individual in a population. By evolving the population, GA finds an optimal collision-free path for the robot. If the physical constraints and uncertainties are considered in path-finding, a robot represents its state as a configuration and plans its path by searching its configuration space. A configuration of a robot can consist of more detailed information other than the location of the robot, including velocities, heading directions, or speed limits. Since the configuration space can be enormous, a robot can sample the configuration space and infer its optimal path using RRT.

Without an environment map, a single robot can plan its motion based on its on-board sensor data, in order to avoid collisions. The common techniques are based on the potential field [42], the dynamic window [24] and nearness diagram [54]. Potential field based algorithms are extensively studied and widely applied. In this approach,

a robot is applied to an artificial force based on the encountered obstacles, and it adjusts its motion based on the applied force. The dynamic window is defined as a set of velocities that the robot to reach within a predefined time interval. Within the set, a robot searches an optimal velocity that can let it move to its goal without collisions as quick as possible. In the nearness diagram, a robot classifies its local observations into different situations using a decision tree. For avoiding collisions, the robot executed the actions associated with each encountered situation.

Starting from this paragraph, this section will focus on single-robot collision avoidance for dynamic obstacles. When an environment contains dynamic obstacles, an environment map can capture their locations and motion dynamics. In this case, a robot can plan its optimal path by searching its configuration space, e.g., using RRT. This collision avoidance problem can also be solved as the case of static obstacles with an additional dimension of time [25]. In this case, a robot can re-plan its collision-free paths as needed.

On the other hand, avoiding dynamic obstacles based on local observations consists of three steps: *detecting, predicting, and avoiding* [21]. *Detecting* requires a robot to distinguish dynamic obstacles from static obstacles based on the data perceived through its onboard sensors [21, 10, 48]. For *predicting*, a robot predicts the information (or features) describing the motion of detected dynamic obstacles based on the data collected by its onboard sensors. Broadly, predicting the motion of dynamic obstacles can be solved by heuristic-based methods (e.g., always assume that the obstacle moving forward) and learning methods (e.g., supervised learning or reinforcement learning) [102]. For *avoiding*, a robot avoids the detected dynamic obstacles based on their predicted motion and the surrounding environment of the robot.

In a real-world application, a mobile robot may need to avoid collisions with human, such as pedestrians. In this case, a robot can still avoid collisions by following the general framework: *detecting, predicting, and avoiding.* When autonomous ve-

hicles or vehicles driven by humans are considered as obstacles, they are generally treated as dynamic obstacles (assuming that they do not react to the robot) in the reviewed literature. In the detecting step, a robot needs to detect humans in its observed environment. For predicting, a robot predicts the motion of detected humans using Kalman filters, social studies about human behaviors [64], or human behavior model estimated offline [2]. In terms of *avoiding*, a robot needs to behave in a socially acceptable way, since a pedestrian may change its behavior when a robot is too close to him [77]. In [40], a robot reasons its velocity space to keep enough distance from pedestrians. However, it is unavoidable for a robot to get close to humans when a robot needs to navigate a crowd of humans. In this case, a robot may not be able to find feasible paths in the step of *avoiding*, although there are narrow spaces among pedestrians allowing it to get through. This issue is termed as the Freezing Robot Problem in [92]. To this end, [92] presents the statistic methods, where a robot cooperates with pedestrians to navigate through a crowd.

2.1.2 Multi-Robot Collision Avoidance

In this section, we review methods for collision avoidance involving multiple robots. In a multi-robot setting, a robot needs to avoid collisions with both obstacles and the other interacting robots. Each robot can treat the other robots as obstacles. Alternatively, the robot can also consider the other robots as components of a multi-robot system, instead of parts of an environment. The application of those two different treatments of multi-robot interaction depend on the amount of mutual information shares by robots.

When an environment map is available to a multi-robot system, each robot in the system can plan a collision-free path based on the map while being mindful of other robots' paths. In a multi-robot system, each robot can have an environment

map or share a global map. Although robots can plan their paths, they will require techniques to minimize interfering with each other during their navigation [35].

In the absence of environment maps, robots can avoid collisions based on their local observation. Collision avoidance in multi-robot systems must allow robots to plan their motion with navigation plans of other robots. Berg et al. [94] proposed reciprocal velocity obstacles for multi-robot collision avoidance. Their algorithm allows each robot to infer actions based on the other robots' velocities and selects an optimal action that would not result in collisions in the near future. Inspired by the velocity obstacles, Chen et al. [16] achieve collision avoidance between two robots by estimating the Q-function for joint states and actions of robots. In their later work [15], their method is generalized for collision avoidance among multiple robots.

The map-based methods for collision avoidance with static obstacles can be extended to deal with the dynamic obstacles, even when there are multiple robots situating in the environment. In this case, each robot can consider the other robots in the system as dynamic obstacles. Mapless collision avoidance is more actively studied in the context of multi-robot systems.

For mapless collision avoidance in multi-robot systems, conventionally a robot can plan its motion based on its observation or the joint observation of all robots. To this end, many existing methods are based on the Reciprocal Velocity Obstacles (RVO) algorithm. The concept of RVO [94] is an extension of velocity obstacles (VO), which is defined as the set of velocities which can make the robot collide with the obstacles or the other robots in the near future. In RVO, when robots encounter each other, each robot takes a certain responsibility for avoiding each other. For example, when the robot r_a and r_b move directly towards each other, VO based methods let both robots assume each other as obstacles and turn to their left 60°. However, with RVO, each of r_a and r_b may take 50% of the responsibility for avoiding each other, so they

only need to turn 30° in this case. The percentage of the responsibility for collision avoidance is predefined and can be used to model the behavior of each robot.

The RVO has been extensively studied and widely applied [80, 95, 3, 70, 8]. In [28], RVO based method which accounts the uncertainties introduced by robot localization is proposed to enable collision avoidance among multiple robots. As an alternative approach, reinforcement learning has also been actively studied for collision avoidance in multi-robot systems. Some of the reinforcement learning approaches [15, 16, 51] are inspired by the VO based approach. That work assumes that each robot can observe the velocities and locations of the other robots. The estimated policy directly maps the observations of a robot to the desired velocity command for collision avoidance. In [44], the multi-agent reinforcement learning is combined with the formation control method, and the learned Q-function is used by each robot to select escaping direction in the context of its neighbors. For obstacles, such as pedestrians, robots must predict their reactions before determining their strategies for collision avoidance.

2.2 Reinforcement Learning

Reinforcement learning requires the interaction between a robot and an environment to be the Markov Decision Process (MDP). The MDP is defined as a tuple (S, A, T, R). S defines a state space of a robot, and A is a set of actions the robot can take in each state $s \in S$. $T : S \times A \times S \rightarrow \mathbb{R}$ is a transition function, define a probability distribution over all possible states resulting from an action taken in any state. $R : S \times A \rightarrow \mathbb{R}$ is a reward function. For each $s \in S$, R assigns a scale value to a robot as a reward. An environment where a robot situates is Markovian if the future states of the robot only depend on the present state. Maintaining the Markovian property of an environment is essential for a reinforcement learning algorithm working correctly.

2.2.1 Preliminaries

In reinforcement learning, a robot takes actions in discrete-time and estimates a parameterized policy μ to determine the strategy of taking action in each $s \in S$. μ can be stochastic or deterministic. In the stochastic case, $\mu : S \times A \rightarrow \mathbb{R}$ defines a probability of a robot taking action $a \in A$ in $s \in S$. As for the deterministic case, $\mu : S \rightarrow A$ directly maps each $s \in S$ to an action $a \in A$. An optimal policy μ^* lets a robot take an action that gives the maximum expected future reward, s.t.

$$\sum_{t=0}^{\infty} \gamma^t r^t$$

. $\gamma \in [0, 1)$ is a constant discount factor and r^t is the reward a robot receives at time t. For learning μ^*, a value-based approach allows a robot to determine μ^* using an optimal value function:

$$V^*(s) = \max_{a \in A} \sum_{s'} T(s, a, s')(R(s, a, s') + \gamma V^*(s'))$$

With V^*, a robot determines the optimal action based on the value of the state resulting from taking action. This requires the environment model T is known to the robot. Therefore, a value-based approach based on V^* is model-based. Alternatively, a robot can determine μ^* using an optimal Q-function:

$$Q^*(s, a) = \sum_{s'} T(s, a, s')(R(s, a, s') + \gamma \max_{a' \in A} Q^*(s', a'))$$

The q-value of a state-action pair $Q(s, a)$ defines the expected future reward that a robot can receive by taking a in s. A value-based approach based on Q^* is known to be model-free. In this case, a robot can select the optimal action a^* based on the values of actions available in a given state without relying on an environment model,

s.t.

$$a^* = \underset{a \in A}{\operatorname{argmax}}(Q^*(s, a))$$

In contrast to the value-based approach, policy gradient methods directly optimize a parameterized policy with respect to its expected reward. Let μ_θ let a policy with a parameter vector θ. If μ_θ is stochastic, it can be optimized based on policy gradients given by

$$\nabla_\theta J(\mu_\theta) = \mathbb{E}\{\nabla_\theta \log \mu_\theta(a \mid s) Q(s, a)\} \tag{2.1}$$

where ρ^{μ_θ} denotes the probability distribution that a robot visits each $s \in S$ using μ_θ.

When μ_θ is deterministic, a robot can optimize it using deterministic policy gradients [79] given by

$$\nabla_\theta J(\mu_\theta) = \mathbb{E}\{\nabla_\theta \log \mu_\theta(s) \nabla_a Q(s, a) \mid_{a=\mu_\theta(s)}\} \tag{2.2}$$

$\nabla_\theta \log(\mu_\theta(s))$ is a Jacobian matrix where an entry in row i and column j represents the gradient of the ith parameter for the jth action. $\nabla_a Q(s, a)$ is a vector of gradients with respect to the Q-function for the action selected by μ_θ in a state. This thesis uses Q-functions to estimate policy gradients, and alternative estimations of policy gradients can be found in [74].

In policy gradient methods, a Q-function can be unknown to a robot initially. In this case, a robot must fit its Q-function based on its state-action trajectories sampled from an environment, while evaluating policy gradients according to equation 6.1 or 7.1. A method alternating between fitting a Q-function and optimizing a policy is called an actor-critic method.

2.2.2 Multi-Task Reinforcement Learning

Multitask reinforcement learning allows a robot to learn a single policy for handling multiple related tasks. One would expect that the tasks to be learned are simple tasks. Through multitask reinforcement learning, a robot can asymptotically reach optimal performance on solving any of the learned tasks. It is not always desired to learn a policy that can handle all the tasks at the same time. Besides integrating knowledge from different sources, a robot should also be able to generalize or transfer the knowledge learned from previous tasks for solving new tasks.

In multitask reinforcement learning, a single robot learns multiple tasks, either sequentially or simultaneously. In terms of learning multiple tasks sequentially, much work has been studied under the topics of Lifelong Learning [4, 88], and Curriculum Learning [22, 53]. Recently, simultaneous multitask reinforcement learning has been actively studied, and impressive results have been achieved in discrete environments [20, 29]. On the contrary, multitask reinforcement learning in continuous environments are less focused, and recent work on these topics is [17, 100].

For learning a multitask policy, Deisenroth et al. [17] studied multitask reinforcement learning in the context of robotics for continuous control. In their work, a multitask policy is represented as a function of robot states and tasks, and, at each optimization step, a single robot optimizes the policy using the policy gradients averaged overall tasks. Yang et al. [100] proposed multi-DDPG, which learns simple robotic control tasks with multiple DDPG actors. With a single shared critic, multi-DDPG learns tasks, specific actors, for each continuous control task.

Using regularization to enable multitask learning has been studied in recent literature. Teh et al. [87] applied γ-discounted KL divergents to task-specific policies in order to simultaneously distill them into a central policy. To learn tasks with different reward scales in parallel, Hessel et al. [29] regularize the gradient update of a shared value function by applying PopArt normalization [97] to task-specific state

values. This thesis utilizes Dropout regularization for learning multitask policies and proposed a multitask reinforcement learning method, Asyn^{d} DDPG. Compared to the aforementioned methods, Dropout regularization is a simple but effective technique for enabling asynchronous multitask reinforcement learning. It avoids having need of task-specific information (e.g., task IDs) for synthesizing knowledge learned in individual tasks into a meta policy. This sheds light on a better direction for learning a multitask policy.

2.2.3 Multi-Agent Reinforcement Learning

In multi-agent reinforcement learning (MARL), robots are asked to learn a global policy or individual local policies for solving tasks in cooperative, competitive, or mixed environments. Compared to single-agent reinforcement, reinforcement learning applied to a multi-agent system faces with a unique challenge posed by interactions among robots. The interactions can introduce randomness to the learning process and can make their environment non-Markovian. Consequently, the theoretical guarantees in single-agent reinforcement learning may no longer hold. Ensuring the Markovian property of an environment is essential for a multi-agent reinforcement learning method to learn optimal policies for a team of robots for solving their tasks at hand.

Multi-agent reinforcement learning can be modeled as a Markov Game.

$$(\mathbf{N}, \mathbf{S}, \mathbf{A}, \mathbf{P}, \mathbf{R})$$

$\mathbf{N}$ is a set of robots. $\mathbf{S} = \{S_1, \ldots, S_n\}$ is state spaces of all robots in $\mathbf{N}$. $\mathbf{A} = \{A_1, \ldots, A_n\}$ is action spaces of all robots in $\mathbf{N}$. Let $\mathscr{S} = S_1 \times \cdots \times S_n$ and $\mathscr{A} = A_1 \times \cdots \times A_n$ be the joint state and action spaces of all robots in $\mathbf{N}$. $\mathbf{P} : \mathscr{S} \times \mathscr{A} \times \mathscr{S} \rightarrow [0, 1]$ is a joint state transition function, and $\mathbf{R} = \{R_1, \ldots, R_n\}$ is a set of reward functions

for each robot in $\mathbf{N}$, and $R_i : \mathscr{S} \times \mathscr{A} \rightarrow \mathbb{R}$ is the reward function of robot i in $\mathbf{N}$. In a Markov game, a environment is assumed to be Markovian, and robots take their actions at the same time. In the above definition, robots are assumed to have complete observation of an environment state.

Broadly, approaches to MARL can be categorized as the Independent Learner (IL) approach and the Joint Action Learner (JAL) approach [12]. In IL, each robot independently learns its own policy and value function. As early work based on IL, [86] empirically proved that independent Q-learning robots can converge on policies satisfying Nash-equilibrium in certain conditions. Alternatively, JAL learns a global policy that maps the joint states of robots to their joint actions. In this case, MARL is deduced as single-agent reinforcement learning. Although JAL inherits the convergence guarantees from single-agent reinforcement learning, the joint state and action spaces grow exponentially as the number of robots increases. This poses a great challenge for the existing single-agent reinforcement learning (e.g., deep reinforcement learning) methods to learn a general policy for controlling multiple robots simultaneously.

Recently, Lowe et al. proposed MADDPG [52], which follows the framework of centralized learning with distributed execution. Lowe et al. evaluated the effectiveness of MADDPG in various simple environments, where decentralized single-agent reinforcement learning methods [57, 50, 55] failed to learn. However, the application of MADDPG to real-world settings was not studied. In practice, robots typically observe an environment through sensors, and it is common for robots to have a high-dimensional observation. This thesis applies MADDPG to real-world settings and learns individual policies for robots with high-dimensional observation in cooperative environments.

Relying on deep neural network representation of value functions, recent literature [82, 67] addressed multi-agent learning with value-based methods. Sunehag et al. [82]

learned q-functions based on linearly decomposed team rewards. Later, Rashid [67] extended this work and learned individual q-functions by decomposing a factorized joint q-function in a non-linear way. In contrast to [82, 67], the proposed method learns a separate q-function directly based on agent-specific rewards and allows the q-function to determine the q-values in the condition of joint observation and actions.

2.2.4 End-to-End Reinforcement Learning

In end-to-end reinforcement learning, a robot learns to map its sensor observation to its actuation commands. It greatly simplifies the design of a robot control while still enabling the robot to have optimal performance. As sensor observation and actuator commands pose enormous state and action spaces, finding exact mapping can be infeasible. In recent literature, end-to-end reinforcement learning approximates the mapping through deep neural networks, as they are commonly considered as universal function approximators in principle.

End-to-end reinforcement learning studies were mostly in single-agent domains [57, 50, 27, 104, 84]. It was made popular by Mnih et al. [57] for learning Deep Q-Networks (DQN) to play Atari games at a superhuman performance. Later, Lillicrap et al. [50] applied end-to-end reinforcement learning to continuous control. The authors proposed the DDPG algorithm, which learns deterministic policies based on deterministic policy gradients. DDPG uses slowly evolved target neural networks to stabilize learning. DDPG was able to learn effective policies in various robot continuous control tasks in simulation. By extending DQN, Hausknecht et al. [27] proposed Deep Recurrent Q-Networks (DRQN) for learning end-to-end policies in partially observable environments. The authors demonstrate that recurrent neural network was able to enable a robot to overcome its partial observability and learn optimal policies based on local observation. The aforementioned work are in simulated

environments, and they showed the potential of applying end-to-end reinforcement learning to practice.

Zhu et al. [104] learned end-to-end policies for visual-based indoor navigation. The authors proposed an actor-critic method to learn a policy that directly maps the image sensor data to actions. Utilizing pre-trained image embedding networks, they were able to generalize the policy learned in simulation to the physical environment and allows a robot to navigate to a target only based its visual input. For mapless collision avoidance, Tai et al. [84] extended DDPG for an asynchronous setting and learned a deterministic policy that maps range sensor data to steering commands. The authors learned the policy in a simulated environment and were able to transfer the learned policy to a real-world environment for safe navigation. However, their work is based on low dimensional range sensor data (i.e., ten range measures are perceived at each sensor scan). For practical deployment, end-to-end reinforcement learning should be able to incorporate high dimensional sensor data.

Extending the success of single-agent end-to-end reinforcement learning to a multi-agent setting faces a great challenge. The interaction among robots makes an environment non-stationary. This issue was actively studied in [43, 16, 51, 45, 23]. However, the existing methods either are direct extension of single-agent reinforcement learning [51, 26] or focus the scenarios where each reinforcement learning agent has small state and action spaces [52, 45, 23]. Applying those methods to robots with higher-dimensional state and action spaces is still an open problem. In this case, learning optimal policies quickly becomes intractable, since the search space of joint policies grows exponentially as the number of robots increases.

Chapter 3

Map-based Collision Avoidance in Navigation Tasks

Map-based collision avoidance requires robots to plan collision-free paths based on an environment map. However, an environment map is usually not available to the robots in advance, and the robots must build the map through environment exploration. This chapter studies environment exploration with multiple robots in simulation. While exploring an environment, robots need to plan collision-free paths based the environment map that they progressively build.

Environment exploration studied in this chapter is essentially navigation tasks involving multiple robots. With the growing popularity of the internet of things (IoT) and cloud computing technologies, modern robotic devices can communicate with each other either globally or locally. In this chapter, we study environment exploration under both centralized and distributed settings, which are defined based on the communication ranges of robots. Through the study, this chapter demonstrates that safe navigation requires both map-based and mapless collision avoidance approaches. As the environment map can not accurately present all possible obstacles, the robots

follows pre-planned paths must use the mapless approach to avoid unforeseen obstacles based local sensor observation.

3.1 Centralized Environment Exploration

This section studies multi-robot environment exploration under a centralized setting. In the exploration task, robots need to build map environments and collect discovered targets. We assumed that a centralized device integrates sensor data of all robots into a consistent map and plans collision-free paths for each of the exploring robots. Specifically, we simulate the target collection behavior of a robot as navigation from the location where a robot discovers a target to a dedicated location in the same environment (i.e., a home base).

This section extends the frontier-based exploration algorithm proposed in [11] to simultaneously explore the unknown environment and collect discovered targets with multiple robots. During exploration, the discovered targets require immediate harvesting [47]. In this case, robots should be allocated efficiently, so that the tasks of environment exploration and target collection can be well balanced.

Task allocation was effectively applied to multi-robot exploration by other researchers [62, 98]. However, among those works, balancing the performance of target collection and environment exploration did not receive enough attention. The proposed work provides insights on how a multi-robot exploration strategy can get benefit from robot target-collection behavior. In extensive simulation runs, we evaluated the collision-free paths planned by robots based on their environment map, according to: 1) their total travel time, 2) the total length of their paths, 3) and exploration redundancy. We compared the proposed work with the frontier-based exploration algorithm proposed in [11] and demonstrate that our work can better balance the tasks of environment exploration and target collection.

3.1.1 Problem Statement

We let n robots explore an unknown environment consisting of m targets and k obstacles. The targets in the environment are randomly distributed, and the obstacles are circular regions preventing robots from entering. We assume that the robots can communicate and exchange information globally and can connect to a central device (e.g., a cloud service). The central device represents an environment as a grid-based map and plans a robot path as a sequence of way-points on an environment map.

3.1.2 Solution Approach

In our foraging task, robots need to simultaneously explore the entire environment while continuously collecting all discovered targets. To effectively fulfill the task, the proposed work extended the multi-robot exploration algorithm proposed in [11]. In contrast to the original algorithm, the proposed algorithm is integrated with an auction-based task-allocation method, which can balance tasks of environment exploration and target collection. For clarity, this section first explains the multi-robot exploration algorithm that has been extended and then introduces how it is integrated with the auction-based task-allocation method.

Coordinated Multi-Robot Exploration with Frontiers

During exploration, a global occupancy map[89], which represents the environment, is shared and maintained by all robots. Since robots are assumed to have accurate sensing capability, the occupancy probability $P_o(c_{i,j})$ of the cell $c_{i,j}$ within the sensor range (the probability that a cell with row index i and column index j in the occupancy map is occupied by an obstacle) can be either 0 or 1. In general, $c_{i,j}$ can be in one of three conditions:

open: $P_o(c_{i,j}) = 0$;

occupied: $P_o(c_{i,j}) = 1$;

unknown: $P_o(c_{i,j})$ is not determined.

Robots explore the environment by moving toward frontiers that will provide the best utilities. Frontiers are defined as the open cells lying on the boundaries between explored and unexplored areas [99]. The utility of a frontier f, $U_r(f)$, for robot r, is determined by two factors, the cost for r to reach f and the number of robots moving toward f. Let R_t be a set of robots to choose a frontier cell to move at time t. For each $r \in R_t$, $C_r(c_{i,j})$ is the cost for r to reach the cell $c_{i,j}$.

Before the cost is evaluated for a robot, a copy of the global occupancy map is convoluted with a 5×5 Gaussian filter. After the convolution, the obstacles represented in the map become larger, so that the robot will not move too close to obstacles. With the convoluted copy of the global occupancy map, the cost for each robot moving toward each of its frontiers is evaluated using Algorithm 1.

Algorithm 1 Calculate cost for robots moving toward frontiers

```
1:  function MapCost(R_t)
2:      for each r ∈ R_t do
3:          c_r ← the cell where r locates
4:          for each cell c_{i,j} in the global occupancy map do
5:              if c_r = c_{i,j} then
6:                  C_r(c_{i,j}) ← 0
7:              else
8:                  C_r(c_{i,j}) ← ∞
9:          while there exists c_{i,j}, s.t. C_r(c_{i,j}) = ∞ do
10:             for each cell c_{i,j} in the global occupancy map do
11:                 if c_{i,j} is not unknown then
12:                     C_r(c_{i,j}) ← min_{δ_i,δ_j ∈ {-1,0,1}} {C_r(c_{i+δ_i,j+δ_j}) · (1 + P_o(c_{i+δ_i,j+δ_j}))} + 1
```

Lines 4 to 8 initialize $C_r(c_{i,j})$ to either 0 or ∞, depending on the location of the robot, r. Then $C_r(c_{i,j})$ is updated within the nested-loop from lines 9 to 11. $P_o(i+\delta_i, j+\delta_j)$ is the probability of an obstacle to be at the location $(i+\delta_i, j+\delta_j)$. Line 11 makes sure that the cost for cells within the frontier boundary. After computing

$C_r(c_{i,j})$ for all cells within the frontier boundary, next step is to choose a frontier for each robot to move.

Let $P(c_{i',j'}|c_{i,j})$ be the probability of the cell $c_{i',j'}$ being covered by the robot in the cell $c_{i,j}$. According to [11], $P(c_{i',j'}|c_{i,j})$ is approximated as $P(c_{i',j'}|c_{i,j}) = \max(0, 1 - d_c/l)$, where d_c is the Euclidean distance from $c_{i,j}$ to $c_{i',j'}$ and l is the maximum sensor range of robots. After sensing the environment, each robot in R_t simultaneously updates the $P_o(c)$ of all the cells it covers in the global occupancy map. The robot with the smallest ID in R_t will assign new frontiers for all in R_t using Algorithm 2. To cover the entire environment, robots keep expanding explored areas until no frontiers are left.

Algorithm 2 Assign a new frontier to each robot

```
function AssignFrontiers()
    determine the set of frontiers F from the global occupancy map;
    for all f_i ∈ F, set all utilities U(f_i) = 1
    for each robot r_j ∈ R_t do
        f* = argmax_{f_i ∈ F}(U(f_i) − β · C_{r_j}(f_i))
        set f* to be r_j's frontier
        ∀f_i ∈ F, U(f_i) ← U(f_i) − P(f_i|f*)
```

Balancing Environment Exploration and Target Exploitation

Robots deliver items from targets as they are discovered. As the target discovery continues, many robots will move toward a fewer number of remaining frontiers and cause congestion. Since the number of Explorers is reduced as well, the congestion is reduced. The proposed work uses an auction-based method to allocate robots for discovered targets. Note that neither the number nor the locations of targets are known in advance. Therefore, it is extremely difficult to compute the number of robots for each target before all targets are found. Algorithm 3, however, ensures that each target will be served by at least one robot.

Algorithm 3 Allocate a robot for a discovered target (executed by each robot)

```
function RobotAtAuction()
    while auction is not closed do
        listen to all the other robots
        if discovered a target T then
            become auctioneer
            broadcast T's location (x_T, y_T)
        if received a target location (x_T, y_T) then
            store (x_T, y_T) in r's local memory
            if r is an Explorer then
                bid ← (1 − d_{r,T}/d_max) + α · (1 − n_{Δt}/N)
                broadcast bid
        if r is the auctioneer and
            received the set of bids, B, from all robots in R_t^E \ {r} then
            best_bid = max(B)
            winner ← the bidder of the best_bid in B; // determine the winner
            broadcast winner
        if received winner and r = winner then
            become Worker
            close the auction
```

The robots have two types: Explorers and Workers. They switch the roles appropriately. Let R_t^E be the set of Explorers at time t. Once a target is discovered by a robot, the robot will become an auctioneer and start auction among all the robots in R_t^E. A centralized scheduler ensures that only one auction can be triggered at any time. The auction in Algorithm 3 consists of four steps:

Announce Task: an Explorer becomes an auctioneer after broadcasting a task;

Bid Task: an Explorer submits a bid to the auctioneer after valuating a task;

Determine Winner: the auctioneer determines the winner based bids;

Close Auction: the auctioneer broadcasts the winner and closes the auction.

The robots' bidding behavior influences the results of an auction. In our algorithm, a robot r bids a target T based on two factors: the distance $d_{r,T}$ from r to T and $n_{\Delta t}$, the number of unknown cells explored by r within the last Δt time units. Let N be

the number of cells in the global occupancy map and d_{max} be the maximum possible distance between any robot and any target. The bidding function of a robot is a linear combination of *task valuation* $(1 - d_{r,T}/d_{max})$ and *robot fitness* $\alpha \cdot (1 - n_{\Delta t}/N)$, where α is a constant ranging over $[0, 1]$. Because of the bidding function, a robot that is closer to the target and covers fewer cells recently will bid higher for the announced target. The robots explored more unknown cells tend to have a better chance to explore more unknown cells in the future; these robots should not become Workers easily. Once the auctioneer receives all the bids, it will set the Explorer, which bids the highest value to be the winner of the auction. The winner will become a Worker and move directly back and forth between the discovered target and the base. If the auctioneer does not win, it will become a regular Explorer. Regarding obstacle avoidance, the proposed work employs the method proposed by [32] for all the robots, including Explorers and Workers. Note that Workers can enter the unexplored areas while collecting targets since they know both locations and radius of risk areas in advance.

When exploration is completed, remaining Explorers, $\overline{R^E}$, will also be allocated to transport discovered targets. The proposed work lets the robot with the lowest ID in $\overline{R^E}$ randomly assign each of the robots in $\overline{R^E}$ to one of the discovered targets.

3.1.3 Evaluation

The proposed work is evaluated in simulation and is conducted on a server with an 8-core CPU and 128GB memory. For 100 robots, one instance of simulation takes about two days. Each experiment reports the average performance of 12 simulation instances with standard deviation. The frontier-based exploration algorithm with no auction is evaluated by a fewer simulation runs for each experiment instance, since the algorithm shows similar performance in each simulation given the same number of robots.

This section first investigates how frontier-based exploration performs with different numbers of robots. After that, the effects of the auction-based task allocation on robots' exploration performance are studied.

Frontier-based Exploration with Various Number of Robots

Given a number of robots, the performance of their exploration is evaluated based on three criteria:

Trajectory Length: total length of robot trajectories during exploration;

Exploration Time: total time for robots to complete their exploration;

Redundancy: times of robots repeatedly updating the same cells in the global map.

Both *trajectory length* and *exploration time* are easy to understand, but the *redundancy* deserves more explanation. During exploration, all robots simultaneously update the global occupancy map based on their sensor data. Therefore, a cell's occupancy probability, in the global occupancy map, may be updated by more than one robot at nearly the same time. This is unlike the work proposed in [89, 11], where cells' occupancy probabilities are learned based on all robots' observation. Since it has been assumed that robots have accurate sensing capability, ideally, each cell should be updated only once. However, coordination among robots may sometimes be inefficient; some cells may need to be updated more than once. For evaluation purpose, each deployed robot keep track of the number of unknown cells in the global occupancy map during entire exploration. Let R denote a set of deployed robots, and n_r denote the number of cells covered by a robot r during entire exploration. Formally, exploration redundancy is defined as $N_R - N$, where $N_R = \sum_{r \in R} n_r$ and N is the total number of cells in the global occupancy map (as defined in section 3.1.2).

Figure 3.4 shows the environment to be explored. The frontier-based exploration algorithm is evaluated based on different numbers of robots, including groups with

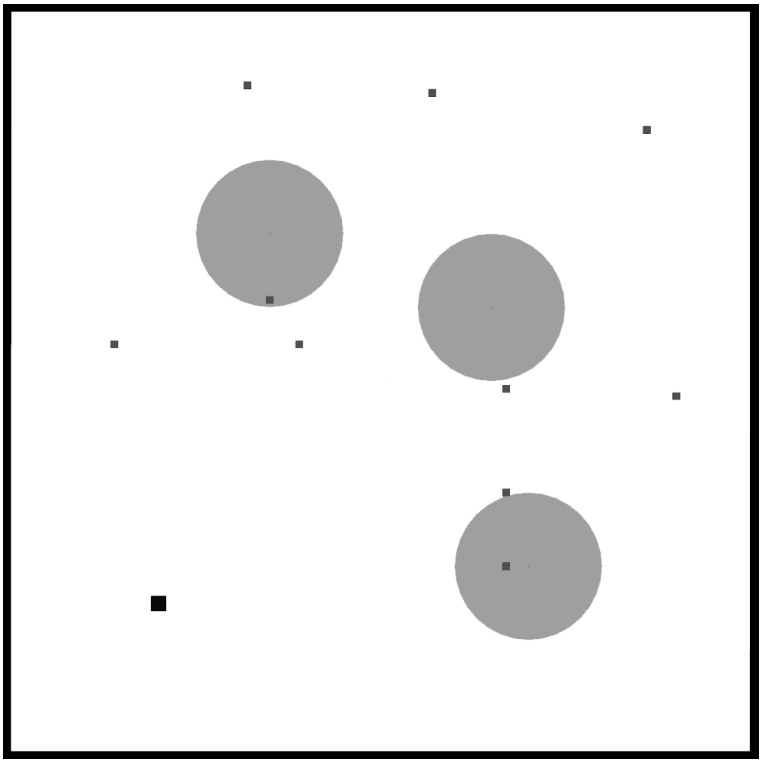


Figure 3.1: The square at bottom left of the figure is the home base and the smaller squares are the targets to be collected. The grey circles represent the risk areas which the robots should avoid.

sizes of 1, 5, 10, 20, 40, 60, 80 and 100. The resulting performance is presented in the left column of Figure 3.2.

Frontier assignment algorithm encourages robots to explore the environment in diverse directions. Consequently, exploration should speed up when more robots are deployed. Figure 3.2-(a) shows the results, where the time of exploration decreases non-linearly as the number of deployed robot increases. Figure 3.3 illustrates an example of the frontier-based exploration, given 20 robots. In Figure 3.2-(a), the time for finishing exploration does not change significantly when the number of deployed robot exceeds 20. On the other hand, Figures 3.2-(b) and (c) show that both trajectory length and exploration redundancy increase as more robots are deployed. The reason is that robots have to avoid colliding with each other more frequently as the environment becomes more "crowd." When robots are more close to each other, the

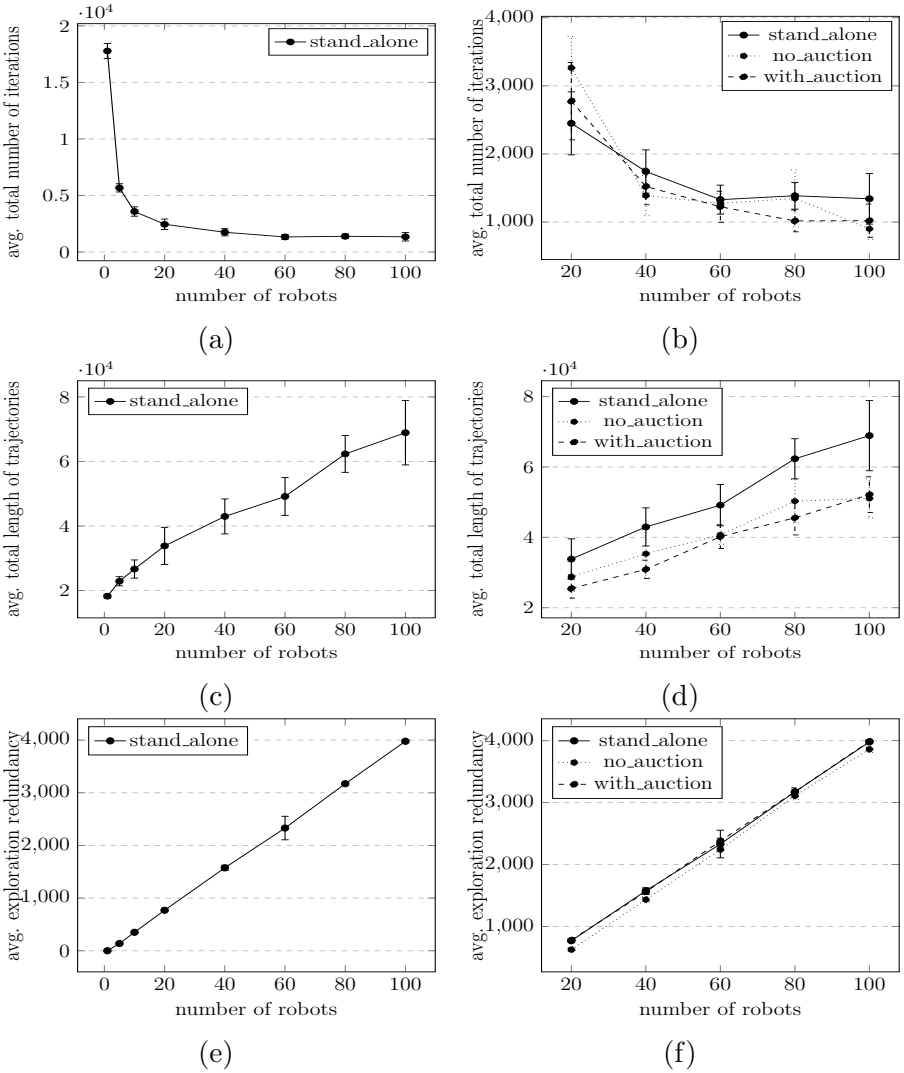
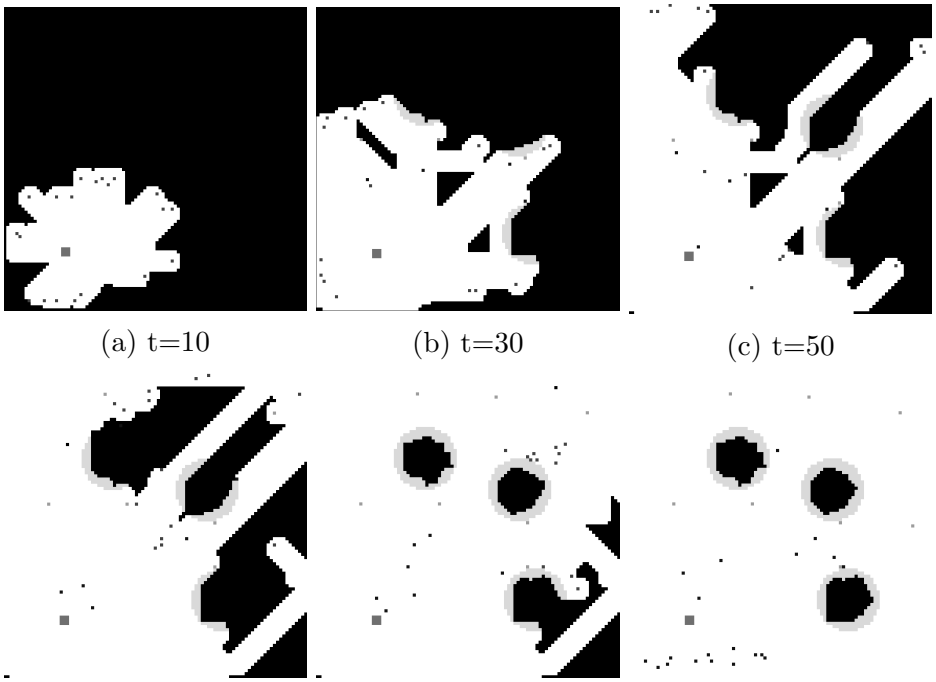


Figure 3.2: Left column: performance of stand-alone frontier-based exploration. Right column: performance of stand-alone frontier-based exploration and frontier-based exploration with target delivery. Standard deviation is presented for the result of each given number of robots.

area covered by a robot's sensor will more likely overlap with the areas covered by the other robots.

The results from frontier-based exploration suggest that there is an optimal number of robots for exploring an environment with unknown size. As the size of unexplored areas shrinks during exploration, the number of exploring robots should also be reduced accordingly, to avoid congestion.



(a) t=10 (b) t=30 (c) t=50

(d) t=70 (e) t=90 (f) t=100

Figure 3.3: The frontier-based exploration with 20 robots. The black areas are unexplored, and the small grey squares are the targets.

Effect of Task Allocation on Exploration

To better understand the effects of the auction, this section presents two series of experiments. The first series allocate the robot without auction, and then the resulting performance is compared with the performance of the proposed auction-based task allocation method.

Task Allocation without Auction

This part of the experiments firstly lets any robot which discovers a target directly become a Worker to collect the target. In contrast to the performance of the standalone frontier-based exploration, the evaluation of the proposed work is based on a different number of robots: 20, 40, 60, 80, and 100. The resulting performance is presented in the right column of Figure 3.2.

Turning Explorers as they discover a target to Workers has limited effects in improving exploration efficiency. Figure 3.2-(d) shows that this approach takes a long time to explore the entire environment when the number of robots is less than 60. If robot density is low, the frontier-based exploration algorithm can well disperse robots. In this case, deploying more robots can speed up the exploration process because explored areas can be expanded in more different directions simultaneously. However, the exploration will be slowed down when some Explorers become Workers. According to the results reported earlier, the total exploration time decreases non-linearly due to congestion. When the number of robots is too high (> 60), having some Explorers become Workers can ease congestion. As a result, total exploration time can be reduced.

Even without auction, as presented in Figure 3.2-(e), the trajectory traveled by robots is significantly decreased when the task allocation method is deployed. Figure 3.2-(f) shows that the exploration redundancy is also slightly decreased due to the task allocation method. The reason is that no matter how many robots are deployed initially, the probability of having an area sensed by multiple robots always decreases as there are fewer Explorers remaining. As a result, overall exploration redundancy is decreased.

Auction-based Task Allocation

In the first series of experiments, the robot discovering a target must be closer to the target than most of the other robots, although the experiments could not guarantee that it is the closest one. Therefore, task valuation is effective for allocating robots. The second series of experiments linearly combine task valuation and robot fitness in robots' bidding functions (as defined in Section 3.1.2). The goal of having robot fitness is to allocate robots that contribute less to the exploration to collect targets. In this series of experiments, the constant α is empirically set as 0.5, and the fitness

of a robot is evaluated based on the number of cells covered by the robot in the global occupancy map within the last 100 simulations iterations. The effect of the auction with both task valuation and robot fitness is shown in the right column of Figure3.2.

In Figures 3.2-(d) and (e), both exploration time and trajectory length are reduced, due to auction. With the bidding function, a robot at a better exploratory position can keep exploring the environment rather than being allocated for target collection. Instead, a robot that is left behind has a higher probability of becoming a Worker, because the areas close to it assigned frontiers are often explored by the robots moving in front of it.

Figure 3.2-(f) shows that exploration redundancy in this series of experiments is almost identical to the one in the stand-alone frontier-based exploration. This is an indication that the proposed auction-based task-allocation method performs well on minimizing the impact of target collection on environment exploration.

3.2 Distributed Environment Exploration

In this section, we study environment exploration with multiple robots under a distributed setting. In this exploration task, robots have limited communication ranges and must build consistent maps of their environment in their local memory. Based on the consistent local maps, robots can plan their own collision-free paths to different way-points, in order to safely explore an environment. To this end, we study communication protocol which enables robots to synchronize their local environment map in an efficient way. Specifically, the communication protocol serves two purposes: 1) enabling robots to form communication networks during exploration, 2) and allowing the robots to synchronize their local environment maps.

Designing an effective protocol, which can meet above purposes, needs to deal with the following challenges. First, robots can quickly exhaust their limited resources after

intensive communication. Second, avoiding redundant information to save communication bandwidth can violate convergence guarantee of a consensus protocol. Third, a communication network formed by robots can dynamically change. It is difficult to ensure network connectivity, particularly in the presence of communication loss [33]. Although the convergence of min, max, and average consensus solutions have been proven under different communication network dynamics [60, 13, 61, 69, 58, 34], the existing consensus-based methods are communication heavy.

Addressing the above challenges, we proposed a consensus-based protocol for build consistent local environment maps in distributed environment exploration. During exploration, robots following the proposed protocol can dynamically constitute connected communication networks and synchronize their local environment maps. The proposed protocol is computationally efficient. It allowed robots to keep track of their communication history and avoid exchanging redundant information. The proposed protocol could preserve its convergence even in the presence of communication loss. In experiments, the proposed protocol is evaluated based on the well-known distributed frontier-based exploration [11]. In extensive experiments, the proposed protocol demonstrated its impact on robot exploration performance and demonstrated its efficiency in terms of its communication demand. Also, the proposed protocol was robust and can enable robots to build consistent maps, even in the presence of communication loss.

Consensus-based algorithms have been studied in the context of distributed multi-robot systems. Aragues et. al [6] proposed an offline method where robots used a consensus algorithm to estimate their global frame of reference for merging their local environment maps. Later, the authors [7] proposed an online algorithm that allowed robots to merge their feature-based environment maps during exploration. In [7], robots estimated their average position through a consensus algorithm for merging their local environment maps. Different from those existing methods, our

work focused on the grid-based environment map. Besides, our work enabled robots to avoid exchanging redundant information and was robust against communication loss.

3.2.1 Problem Statement

This section presents the consensus-based communication protocol for building consistent local environment maps in distributed exploration. Let $\mathscr{N} = \{1, 2, \ldots, n\}$ be a set of robots. Each robot $i \in \mathscr{N}$ has an $n \times m$ grid M_i which represent the environment mapped by the robot. Each cell c_i in M_i corresponds to a square area in the environment where robot i operates, and it is associated with a label $L(c_i) \in \{open, occupied, unknown\}$. Let $M_i(t)$ be the environment map built by robot i at time t. Let $M_i(t)$ be the state $x_i(t)$ of robot i at time t, s.t.

$$x_i(t) = M_i(t)$$

Let $G(t) = (V, E)$ denote a unweighted undirected graph representing a communication network formed by some robots $V \subset \mathscr{N}$ at time t. E is a set of edges, where each $e_{ij} \in E$ indicates that robot i and j can communicate with each other. For each robot $i \in V$, the robot update its state $x_i(t)$ according to a protocol $\mu_i(t)$, s.t.

$$\dot{x}_i(t) = \mu_i(t)$$

Let $x_i(t)$ be a initial state of robot i at time t; $\boldsymbol{x}(t)$ be a vector of $x_i(t), \forall i \in V$. Robots in V reach χ-consensus, if there is a stable state $x^* = \chi(\boldsymbol{x})$, s.t. $\forall i \in V$, $x_i(t + \delta) = x^*$ with $\delta \to \infty$.

3.2.2 Solution Approach

The proposed consensus-based protocol enables robots to share environment maps in distributed exploration. Robots following the proposed protocol can form connected communication networks and synchronize their local environment maps. Robots usually have limited computing resources and may suffer communication loss in practice. The consensus-based protocol is designed to let robots keep track of their communication history and avoid communicating redundant information. Besides, the proposed protocol can preserve its convergence guarantee in the presence of communication loss.

Let $G(t) = (V, E)$ be a connected graph representing a communication network constituted by a set of robots $V \subset \mathscr{N}$ at time t. For each $i \in V$, let $S_i^{kn}(t)$ be a set of cells known to robot i, s.t.

$$S_i^{kn}(t) = \{c_i \mid c_i \text{ is a cell of } M_i(t) \text{ and } L(c) \neq unknown\}$$

Let $N(i)$ be the neighbors of robot i; h be a function, s.t.

$$h(c_k, k) = \begin{cases} 1, & \text{if } c_k \in S_k^{kn}, \forall k \in N(i). \\ 0, & \text{otherwise.} \end{cases} \tag{3.1}$$

To avoid broadcasting redundant cells, at time t, an robot i will broadcast a set of cells $S_i^{st}(t + \delta)$, s.t.

$$S_i^{st}(t) = \{c_i \mid c_i \in S_i^{kn}(t) \text{ and } \exists k \in N(i),\ h(c_i, k) = 0\}$$

Intuitively, $S_i^{st}(t + \delta)$ contains all the cells in $M_i(t)$ that haven't been sent to all the neighbors of robot i. Let δ be the time that have passed after the initial time t. At

time $t+\delta$, each $i \in V$ updates its state $x_i(t+\delta)$ according to the consensus protocol

$$\mu_i(t+\delta) = l(\bigcup_{k \in N(i)} S_k^{st}(t+\delta)) \tag{3.2}$$

where l is a function which updates $M_i(t+\delta)$ based on the cells robot i has received from its neighbors.

Theorem 1. *Let $G(t+\delta) = (V, E)$, $\forall \delta \in \mathbb{N}$, be a connected graph with node set V and edge set E, where t is the time when G is initially formed and δ is the passed time steps after t. Suppose $G(t+\delta), \forall \delta \in \mathbb{N}$, has fixed topology, and there is no communication loss among robots. As $\delta \to |V|$, protocol 3.2 can have all $i \in V$ reach χ-consensus, where each $M_i(t)$ is merged with $M_j(t), \forall j \in V/\{i\}$.*

Proof. The convergence of protocol 3.2 can be proved by contradiction.Besides, the proof also derives the upper bound of δ for robots to reach consensus. $\mathscr{S} = \bigcup_{\forall i \in V} S_i^{kn}(t)$ is the total possible known occupancy probabilities that can be exchanged by all robots over the period of δ. According to the update function l, the number of known cells in robot i's environment map $M_i(t+\delta)$ is monotonically increasing with respect to δ. Here, a robot $i \in V$ is considered to have a cell $c \in \mathscr{S}$, if the cell c is known in $M_i(t+\delta)$.

Suppose $S_i^{kn}(t+\delta) \subset S^{kn}$, as $\delta \to \infty$. In this case, let j be an robot that can be reached by i, and j does not have a cell c. According to the definition of l, all the robots in $N(j)$ must not have c in their environment map. Besides, the neighbors of robots in $N(j)$ must not have c too. By induction, all the robots in G that can be reached from v_i must not contain c. Since $G(t+\delta)$ is connected and has a fixed topology for all $\delta \in \mathbb{N}$, there is always a path from i to each of the other robots. This contradicts the assumption of $S_i^{kn}(t+\delta) \subset S^{kn}$. Therefore, $S_i^{kn}(t+\delta) = S^{kn}$ for all $i \in V$, as $\delta \to \infty$.

Algorithm 4 Forming connected communication networks

```
function ConstituteNetwork()
    listen for messages from other robots
    if connected to robot k then
        if k ∉ N(i) then
            Stop moving
            N(i) ← N(i) ∪ {k}
        use protocol 3.2 to update M_i(t) based on S_k^st(t + δ)
        broadcast S_i^st(t + δ)
        δ ← δ + 1
    if N(i) = {} or reached consensus then
        continue to explore the environment
        δ ← 0
```

To derive the upper bound of δ, the application of protocol 3.2 is considered as information cascade on $G(t)$. $c_j \in S_j^{kn}(t)$ is propagated from j to i, if i receives c_j from j. At each time step, each robot $i \in V$ propagate all $c_i \in S_i^{kn}(t)$ to all the other robots. For all robots in V, $S_i^{kn}(t)$ are propagated in parallel. Note that, at each time step, each cell c_i in $S_i^{kn}(t)$ can be propagated one hop between two robots, and the propagation stops when a robot has already had c_i. In this case, there is no cycle during the propagation. Because of the parallel propagation, the time for robots to reach consensus is determined by the longest simple path that c_i is propagated through. In $G(t)$, the longest simple path between any two robots has the length of $|V| - 1$. Therefore, the time step for robots in V to reach χ-consensus is bounded by $|V|$. □

To synchronize local maps of robots, the protocol 3.2 requires $G(t + \delta)$ to remain connected for all $\delta \in \mathcal{N}$. To constitute a connected communication network during exploration, each robot in V try to connect to its nearby robots periodically. A robot i stops moving when it connects to another robot, and then it starts broadcasting $S_i(t)$. Let t be the time when robot i joins a communication network. Algorithm 4 presents the protocol for robots to constitute a connected communication network during exploration.

A robot considers other robots in the same communication network reaches consensus, if its own environment map has not been updated for a certain amount of time. Note that, according to protocol 3.2, two robots will broadcast empty sets of cells to each other, if they have already shared all the known occupancy probabilities in their own environment maps.

Theorem 2. *At any time t, the communication network $G(t+\delta), \forall\delta \in \mathbb{N}$, constituted by robots using Algorithm 4 is connected.*

Proof. Based on Algorithm 4, during exploration an communication network is initialized two robots. Given a communication network $G(t)$ consisting of n robots, $n \geq 2$. Suppose $G_0(t)$ is $G(t)$'s initial network consisting of two robots i and j. Let k be the third robot joining $G(t)$ at time $t+1$. In this case, k must be able to communicate with either i or j. Since $G(t+1)$ is undirected, there is a path between any two of i, j and k. Therefore, $G(t+1)$ is connected.

To prove the network connectivity by induction, $G(t+\delta), \forall\delta \in \mathbb{N}$ is defined as a connected undirected graph representing a communication network formed by a set of robots V at time $t+\delta$. Let x be an robot joining $G(t+\delta)$ at time $t+\delta+1$. In this case, x must be able to communicate with at least one robot $i \in V$. Since $G(t+\delta)$ is connected, each $j \in V/\{i\}$ has at least one path to i, and vice versa. Since for each $i \in V$ there is at least one path between i and x, $G(t+\delta+1)$ is a connected graph. By induction, $G(t+\delta), \forall\delta \in \mathbb{N}$ is connected. Therefore, a communication network constituted by robots using Algorithm 4 is always connected. □

Theorem 2 and 1 prove that robots using Algorithm 4 can eventually form a connected communication network with fixed topology, and the proposed protocol can let robots eventually reach a consensus of their environment maps. At last, Theorem 3 prove that the robots can preserve the convergence guarantee in the presence of communication loss.

Theorem 3. *Let $G(t+\delta) = (V, E), \forall \delta \in \mathbb{N}$, be a connected communication network formed by a set of robots V at time t. Suppose $G(t+\delta)$ is a weighted graph, where an edge $e_{ij} \in E$ represents the probability of robot i receiving information from robot j through communication. As $\delta \to \infty$ all robots in V can reach χ-consensus, where each $M_i(t)$ is merged with $M_j(t), \forall j \in V/\{i\}$*

Proof. Theorem 1 proves the convergence of the proposed consensus protocol in the weighted communication network if one can guarantee that each robot in V can eventually receive all the known cells in all its neighbors' environment map. Note that the proposed protocol does not consider the cells whose occupancy probabilities are unknown to all robots. Therefore, those cells will not have any impact on convergence.

Let $N(v_i)$ be a set of neighbors of robot i. Suppose cell c_i unknown in $M_i(t)$ but known by at least one $k \in N(i)$. At each time step, the probability of robot i updating c_i to be known is $\Sigma_{v_k \in N(v_i)} \alpha e_{ik}$, where α is a decision variable. $\alpha = 1$ if c_i is known by k. Otherwise, $\alpha = 0$. According to equation 3.1 and 3.2, each $k \in N(i)$ will repeatedly broadcast c_i, if c_i known by robot k. Given δ time steps, the probability of robot i marking c_i to be known is $\delta \Sigma_{v_k \in N(v_i)} \alpha e_{ik}$. According to the equation, this probability will monotonically increase with respect to δ. Therefore, as $\delta \to \infty$, it is guaranteed that robot i will update c_i to be known based on the cells broadcast by its neighbors. □

3.2.3 Evaluation

The experiments presented in this section integrate the proposed protocol with the frontier-based distributed exploration [11]. The experiments are implemented in C++ using Gazebo [41], and they are executed on a server with 8 CPUs and 128Gb memory. The simulated robots have diameters of 0.5m and communication ranges of 20m. Each robot is equipped with a LiDar sensor that supports 360° sensing with 360, evenly

spaced lasers. Each laser has a range of 0.1m and 10m and consists of Gaussian noises with the zero mean and the standard deviation of 0.1.

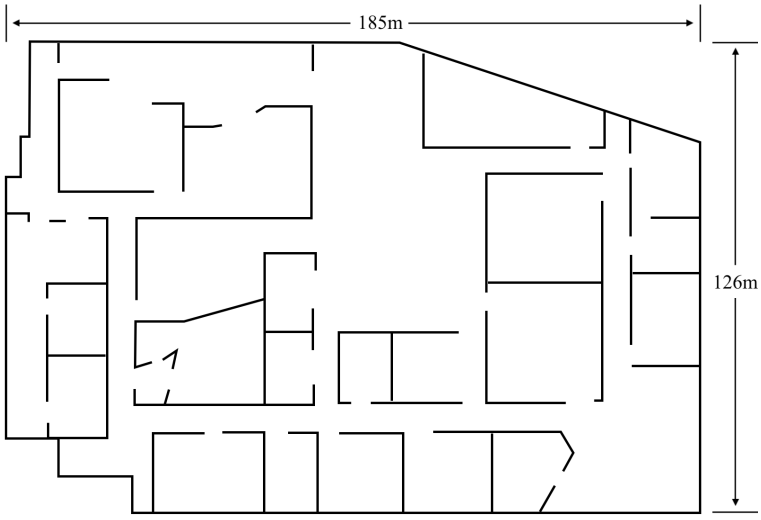
During exploration, a robot uses its LiDar sensor for both environment mapping and collision detection. It has a linear motion model [90] and moves at a constant speed. For collision detection, each robot senses the environment within the corn of 120° in its front at a constant rate. If the robot detects objects that are 0.8m away, it stops moving immediately and then keeps turning to its right until no objects can be detected.

For all the experiments presented in this section, the robots are always simulated with the same configuration. All the methods to be evaluated are empirically adjusted to their best performance. The exploration time and the trajectory length of robots are evaluated by using the simulation time and distance measure provided by Gazebo [41].

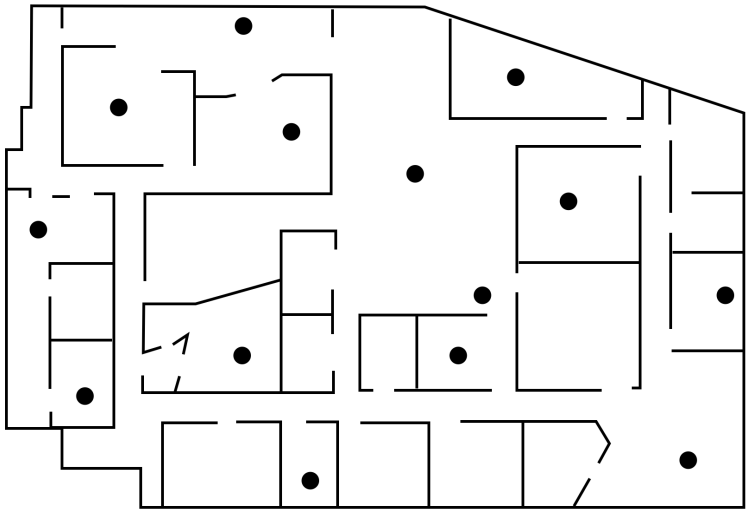
Impact on Exploration Efficiency

The proposed protocol on environment exploration is applied to the frontier-based exploration, where robots always move to their closest frontiers.

Having a consistent view of the mapped environment can be easily achieved when robots have unlimited communication ranges (i.e., each robot can always communicate with all the other robots.). In contrast, the proposed protocol enables robots with limited communication ranges to have consistent local environment maps after reaching consensus. The performance of the frontier-based exploration algorithms with and without the proposed protocol, in order to evaluate the impact of the proposed protocol on exploration. The frontier-based exploration without the proposed protocol assumes that robots have unlimited communication ranges. For each number of robots, the result is based on their average performance in 12 simulation runs, and the error bars are standard deviation.



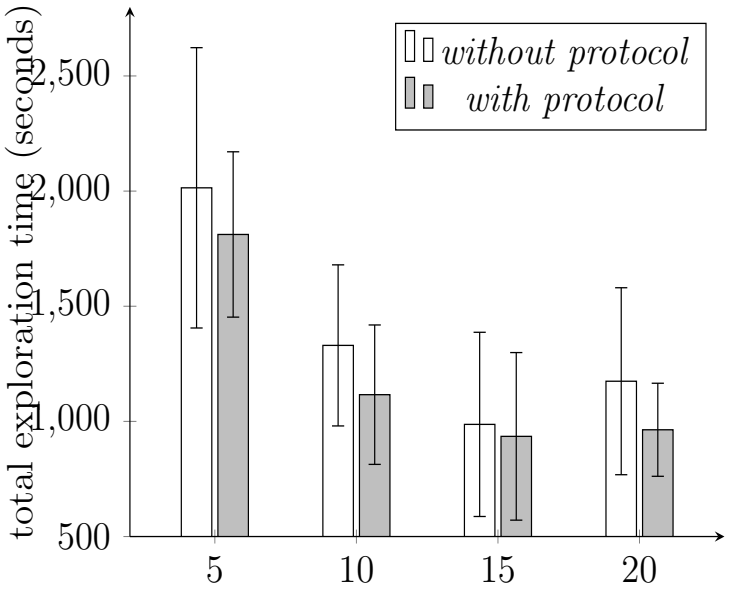
(a)



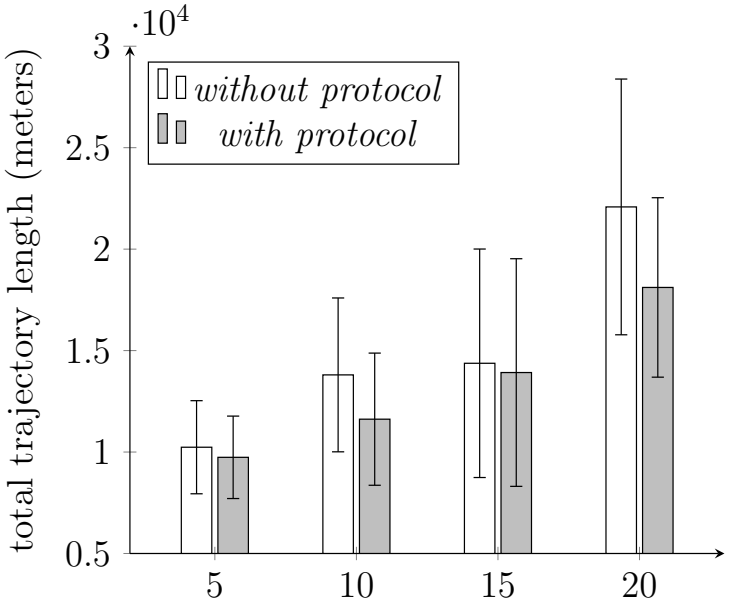
(b)

Figure 3.4: (a): an indoor environment with size of $185m \times 126m$. (b) the indoor environment in with randomly distributed cylindrical obstacles. The solid black circles represent the obstacles.

As presented in Figure 3.5a, the proposed protocol enabled robots with limited communication to conduct faster exploration. The results indicate that the time for robots to reach consensus is reasonable and does not introduce significant overhead. The resulting trajectory lengths show that the proposed protocol can enable robots to complete their exploration with shorter travel distances. However, according to exploration time, robots have decrease exploration performance, when their density



(a) Average exploration time.



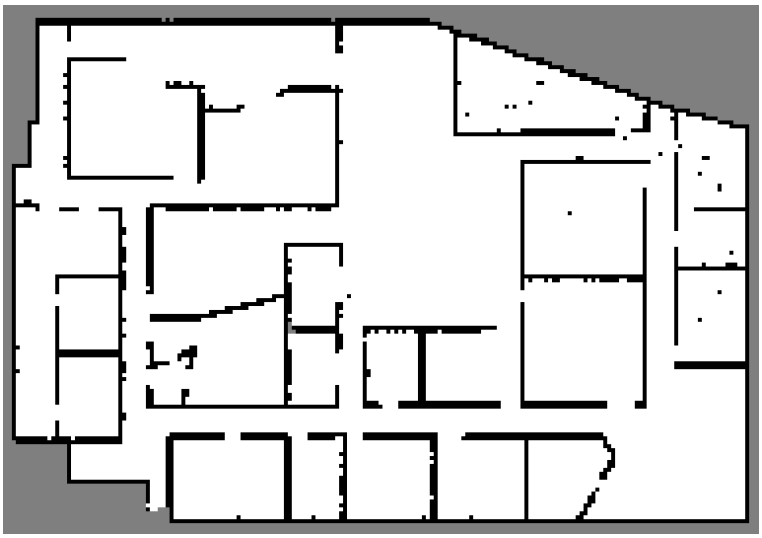
(b) Total length of trajectories.

Figure 3.5: For the environment presented in Figure 3.4.

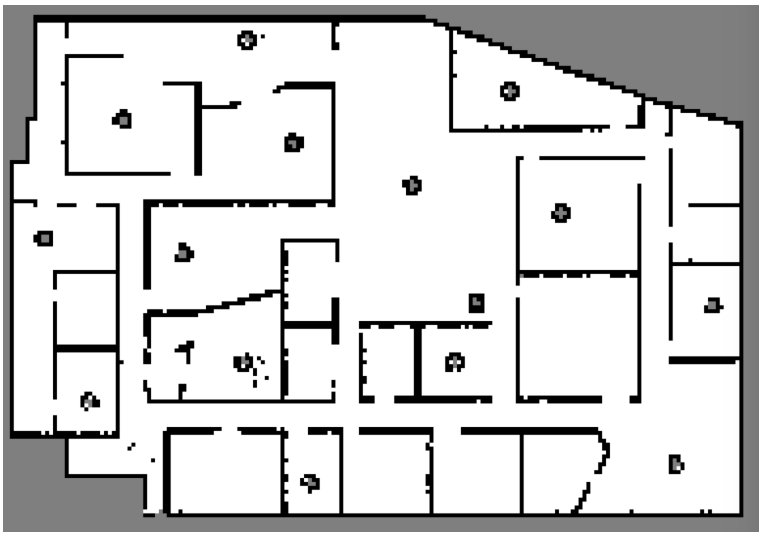
exceeds a certain threshold (i.e., 15 robots). The reason is that, as the density of robots increases, their interference between each other (i.e., robots avoid colliding each other) becomes more significant.

Collaborative Mapping

The proposed protocol must enable robots to build the correct environment map after the exploration. This section lets ten robots map both environments presented in Figure 3.4 using the proposed protocol. Figure 3.6 presents the resulting maps of both environments at the end of exploration. Each cell in a robot's local environment



(a) The map of Figure 3.4a.



(b) The map of 3.4b.

Figure 3.6: The environment maps built by 10 robots following the proposed protocol.

map corresponds to a $1m \times 1m$ area in the environment. In general, robots following the proposed protocol can correctly merge their local environment maps and estimate

the overall maps in both environments. The correctness of the map can be verified by comparing the structure of the mapped obstacles with ground-truth presented in Figure 3.6. Besides the structures of mapped walls, one can clearly identify the randomly distributed obstacles based on their shapes and sizes in Figure 3.6b. With a single LiDar sensor, a robot can not distinguish obstacles from the other robots. The small squares scattered in Figure 3.6 are robots that are mistakenly detected as obstacles. Since sensor data of robots consists of small Gaussian noises, robots can map the same obstacles into slightly different cells in their local environment maps. That causes the mapped obstacles to have non-smooth edges.

Communication Demand

Following the proposed protocol, each robot keeps track of its communication history. This is done by having a robot maintaining a simple data structure to keep track of the cells that have been broadcast to each of the other robots. For the evaluation of communication demand, this section compares the amount of information exchanged by ten robots with and without following the proposed protocol during exploration. The amount of information exchanged among robots is approximated as the total number of occupancy probabilities broadcast by all the robots during exploration. Without following the proposed protocol, each robot in the same communication network broadcasts all the known occupancy probabilities in its local environment map until convergence. The experiment results are presented in Figure 3.7, where the error bars represent the standard deviation. Table 3.1 summarizes the amount of exchanged information, given different number of robots. For each number of robots, the result is based on their average performance in 12 simulation runs.

With the proposed protocol, each robot only broadcasts the cells that have not been sent to all its current neighbors. As presented in Figure 3.7, the proposed protocol can significantly save the communication bandwidth for sharing information.

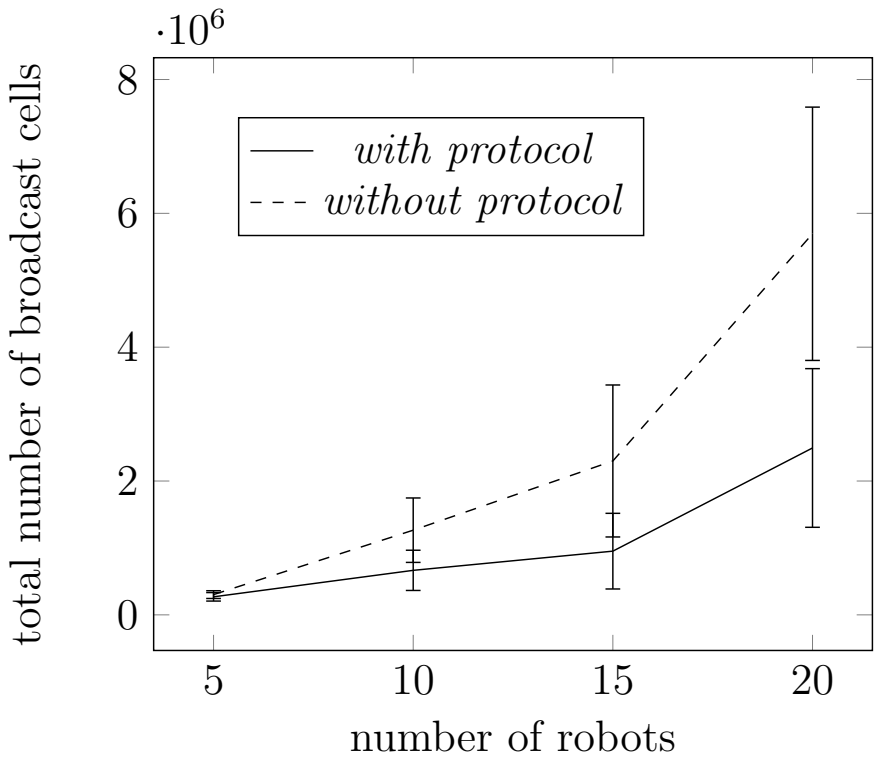


Figure 3.7: The total amount of information exchanged by 10 robots with and without following the proposed protocol in their frontier-based exploration.

number of robots	total broadcast times	
	proposed approach	*closest frontier exploration*
5	251	335
10	690	1116
15	1428	1683
20	2261	3475

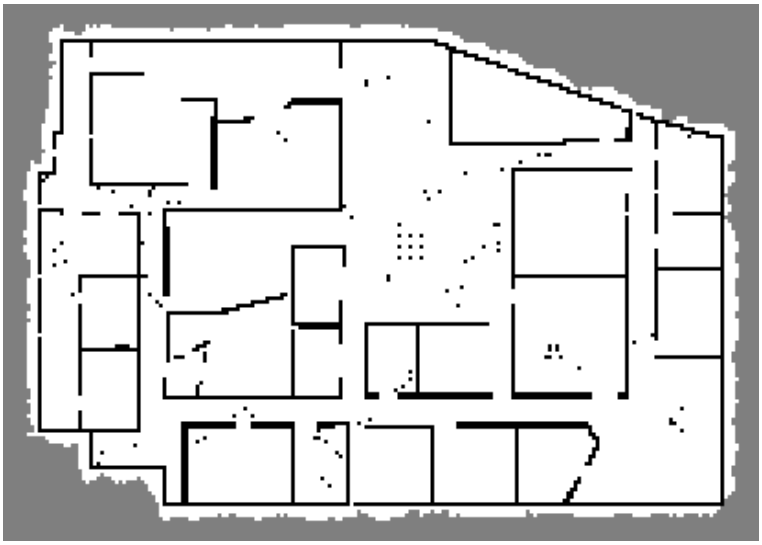
Table 3.1: The total number of times robots broadcast the cells of their occupancy matrices in the proposed approach and in the *closest frontier exploration*.

In the naive approach, a robot has to broadcast all the known cells in its environment map. With the proposed protocol, the number of cells that robots have to broadcast grows mush slower as more robots are deployed for exploration.

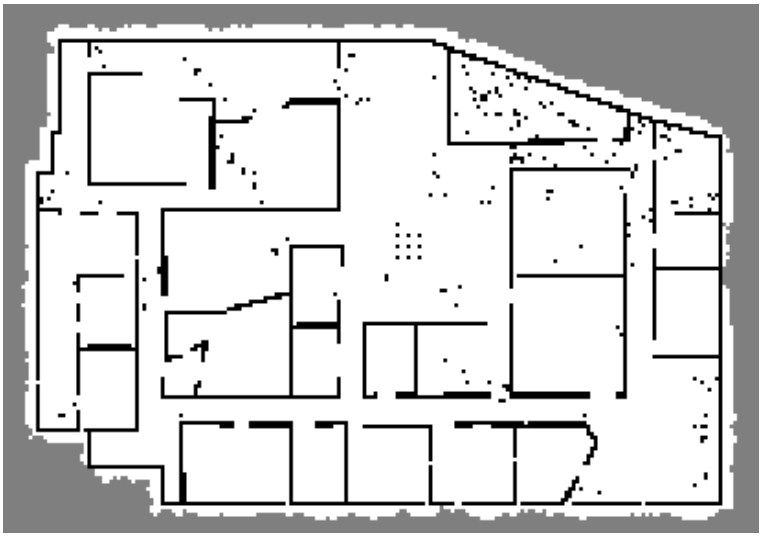
Environment Mapping under Communication Loss

The proposed the protocol is applied to a group of 10 robots with communication loss. As we assumed that all the robots have the same capability, in this part of

experiments, each robot has the same probability of loosing the message broadcast by its neighbors. Specifically, two experiments are conducted. In those experiments, robots have communication loss probabilities of 0.2 and 0.4 respectively. Figure **??** shows final environment mapped by the robots in both experiments.



(a) comm loss: 0.2



(b) comm loss: 0.4

Figure 3.8: The environments mapped by 10 robots with communication loss.

In both experiments, a sufficiently large threshold is set as the maximum broadcasting iterations for robots. The threshold ensures that the robots can converge on their consensus. The results of both experiments suggest that the proposed protocol

is robust against communication loss. As higher communication loss requires larger threshold for broadcasting iterations, robots with higher communication loss have to spend longer time on maintaining their communication network during exploration. In this case, each robot can have a higher probability of being falsely detected as an obstacle by other exploring robots. Therefore, in Figure 3.8, the map built by robots with the communication loss probability of 0.4 is noisier.

3.3 Summary

In this chapter, we studied map-based collision avoidance in multi-robot exploration under both centralized and distributed settings. Environment exploration studied here essentially is multi-robot navigation tasks. For safe navigation, robots need to build environment maps and plan collision-free paths during exploration. However, as demonstrated in extensive experiments, robots following the pre-planned paths can encounter other moving robots or obstacles that are inaccurately represented on the map. In this study of map-based collision avoidance, when a robot encounter an unforeseen obstacle, it will stop moving immediately and then turn to a predefined direction until it can safely move forward. Although this simple mechanism allowed the robot to avoid collisions, it greatly decreased the navigation performance of the robot. Besides, it could easily cause congestion during exploration.

For optimal navigation performance, robots must adopt more effective mapless collision avoidance methods. For mapless collision avoidance, a robot needs to plan its motion based on its local sensor observation. Conventional approach formulate mapless navigation as a real-time optimization problem and require a robot to select for an optimal action based on on-board sensor data within a predefined time limit. The conventional methods usually require complex motion and sensing models of a robot and also requires extensive mutual information among robots. Beside, deploying

those methods generally involves extensive engineering effort and parameter tuning for different environments. Addressing those issues, the rest of the thesis studies the reinforcement learning approach for mapless collision avoidance. The proposed reinforcement learning approach can avoid the need of online action searching and allows robots to learn general policies for mapless collision avoidance in different scenarios under real-world settings.

Chapter 4

Mapless Collision Avoidance through Reinforcement Learning

In the map-based approach studied in the previous chapter, an environment map could contain high-level uncertainties. Following a path pre-planned based on the map, a robot may encounter unforeseen obstacles during navigation. For safe navigation, a robot needs mapless collision avoidance to plan motion based on local sensor observation.

Mapless collision avoidance is a long-standing and open-ended problem in robotics research. Various approaches have been explored under different robot configurations. This thesis studies the reinforcement learning approach for mapless collision avoidance. To this end, the problem of mapless collision avoidance is formulated as Markov Decision Process (MDP). Reinforcement learning is mostly studied in artificial domains. Applying reinforcement learning must deal with the following issues, including partial observability [84], high-dimensional sensor data [51], real-time control delay [19], multi-task environments [36], and multi-agent interaction [37]. Those issues limit the applications of reinforcement learning in real-world environments, and the existing approach can only partially address them. In this thesis, we address

all the above issues and apply reinforcement learning to mapless collision avoidance under real-world settings. This chapter formulates mapless collision avoidance as a reinforcement learning problem and presents details of both robot model and policy representation which will be used in the rest of the thesis.

4.1 RL for Real-Time Continuous Control

In real-world environments, robots operate in real-time. It is common for a robot to have actions consisting of continuous actuator signals (e.g., desired linear and rotational velocities). As actuating an action can take various amounts of time, a robot must meet a predefined time limit (i.e., a real-time constraint) while avoiding collisions.

A robot operating in real time needs to complete the following procedures at each time step: processing sensor data, inferring actions, and actuating commands. It unavoidable for the robot to have time delay before actuating its action. We define the time delay before a robot actuates an action at each time step as control delay. With the aid of modern computing architectures (e.g., GPUs), the robot can process sensor data with minimal delay and transmit control commands to actuator quickly. However, the procedural of inferring actions may require a robot to search a large action space. Besides, this procedure may also requires the robot to optimize its controller based on its previous actions. Therefore, completing the procedure of inferring action can take varying amount of time, particularly when a robot has limited computing resources. Therefore, this procedure can cause high variance of control delay. In Chapter 5, we show that the high variance of control delay can destabilize reinforcement learning for a robot operating in real time and make an environment Non-Markovian.

4.2 MDP Formulation

Let $\mathbf{N} = \{1, \ldots, n\}$ be a set of robots, and e be an environment where the robots operate. Let e be a continuous 2D plane which contains a set of obstacles. Each obstacle occupies an certain region of e. It is assumed that each robot $i \in N$ can accurately determine its current location and follows an pre-planned path $\tau_i = (\rho_1, \ldots, \rho_m)$, where $(\rho_1, \ldots, \rho_m)$ is a sequence of accessible locations on e. To follow τ_i, robot i must visit each ρ_i consecutively until finally reaching ρ_m. Given a pair of ρ_i and ρ_{i+1}, a robot treats ρ_{i+1} as its goal location g while conducting mapless collision avoidance. Following the path τ_i, an robot can come across obstacles or other robots. Therefore, each robot $i \in N$ must dynamically plan a collision-free path from ρ_i to ρ_{i+1}. Specifically, this section focuses on the mapless collision avoidance problem, in which each robot $i \in N$ plan its local collision-free paths from ρ_i to ρ_{i+1} in order to follow its trajectory τ_i.

The mapless collision avoidance is formulated as a 5-tuple $(\mathbf{N}, \mathbf{S}, \mathbf{A}, \mathbf{R}, \mathbf{T})$, where

- $\mathbf{N} = \{1, \ldots, n\}$ is a set of robots situating in the environment e.
- $\mathbf{S} = \{S_1, \ldots, S_n\}$, where S_i is the state space of robot $i \in N$.
- $\mathbf{A} = \{A_1, \ldots, A_n\}$, where A_i is the action space of robot $i \in N$.
- $\mathbf{R} = \{R_1, \ldots, R_n\}$, where R_i is the reward function of robot $i \in N$, such that, $R_i : S_i \times A_i \to \mathbb{R}$
- $\mathbf{T}$: a transition function, such that, $T : \mathscr{S}^t \times \mathscr{A} \to \mathscr{S}$, where $\mathscr{S} = S_1 \times \cdots \times S_n$; $\mathscr{A} = A_1 \times \cdots \times A_n$.

In the above formulation, each robot $i \in \mathbf{N}$ as a function $i : S_i \to A_i$ and assume that each robot independently follows a Markov Decision Process (MDP). At each time step, all robots take deterministic actions at the same time. For collision avoidance,

robots take joint actions which resulting in maximum expected future rewards in their joint states, s.t. $\Sigma_{t=0}^{\infty}\gamma^t\Sigma_i R_i(a_i^t, s_i^t)$, where $a_i \in A_i$, $s_i \in S_i$.

In MDP, a robot operates in discrete time steps. In practice, that requires a robot to discretize continuous time with a specific resolution. Let Δt denote the duration of a time step, and Δt is also the time constraint for a robot to both determine its action based on the current state and measure the state resulting from its action.

4.3 Robot Model

For each robot $i \in N$, its state $s^t \in S_i$ at time t is defined as $(\bar{o}^t, p^t, g)$. $p^t \in \mathbb{R}^2$ is robot i's geographical location. $g \in \mathbb{R}^2$ is the goal location of the robot, which remains fixed during robot navigation. $\bar{o}^t = (o^{t-l+1}, \ldots, o^t)$ represents a sequence of its observation perceived through on-board sensors (e.g., LiDar sensors or cameras), where l is the length of the sequence. By using a temporal sequence of sensor observation, a robot can have improved observability [27] in a partially observable environment. An action of robot i is a vector of real values, s.t. $a^t = (v^t, \omega^t)$, $\forall a \in A_i$. $v^t \in \mathbb{R}^+$ specifies a target speed at a robot's heading direction, and $\omega^t \in [-\pi, \pi]$ is a target rotational velocity of the robot. This robot model closely matches the ROSbot configurations. However, it can be also generalized to model other robots with equivalent sensors and actuators.

A robot perceives $\bar{o}^t$) and p^t through an on-board range sensor and a SLAM-enabled device. At the time t, the range sensor enables robots to perceive o^t as a a vector of normalized range data, s.t. $o^t \in [0, 1]^n$. Each component of o^t represents the ratio between the robot's distance to a detected obstacle and the robot's maximum sensing range. A component with a value of 1 indicates that no obstacle is detected. Figure 4.1 illustrates an example of a robot perceiving o^t at time t through its range sensor.

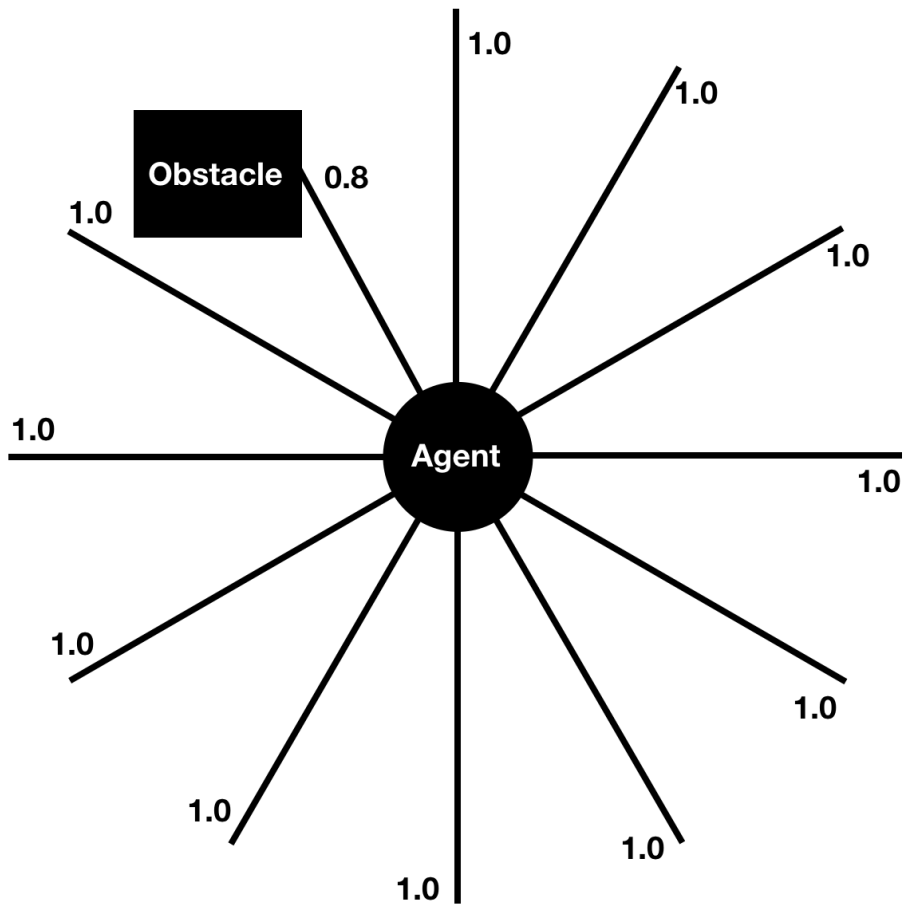


Figure 4.1: An example of robot sensing.

A robot determines its geographical locations $p^t \in \mathbb{R}^2$ through a SLAM-enabled device. The robot is assumed to be a SLAM-enabled device and be able to determine its geographical locations accurately. Having accurate localization is not a strong assumption since the modern SLAM algorithms can provide high-quality state estimations. The actuator of a robot takes a velocity command (v^t, ω^t) as input and accelerates the robot's current v and ω towards the target v^t and ω^t for ϵ time. Let a_v and a_ω be the acceleration for the linear and rotational velocities of the robot. The actuator changes the velocity of the robot according to the follow equation: $v^t = v^c + a_v \cdot \epsilon$, $\omega^t = \omega^c + a_\omega \cdot \epsilon$.

4.4 Neural Network Design for MCA

With reinforcement learning, a robot avoids collisions by mapping its state space to action space through a parameterized policy. The mapping between the state and action spaces is highly nonlinear and can have intricate patterns. To accommodate the complex mapping, we represent the policy of a robot as a deep neural network. Specifically, at time t, the neural network takes a robot state $(p^t, \bar{o}^t, g)$ as input, and output the action (v^t, ω^t) consisting of the target velocities of the robot. With limited sensor ranges, the state perceived by the robot is only based on the robot's surrounding environment. This makes the robot have partial observability during navigation. Also, $\bar{o}^t$, as part of the state representation of the robot, is usually high-dimensional. It is crucial to have an effective neural network structure for the robot's policy to deal with both partial observability and high-dimensional sensor data.

As reported by Hausknecht et al. [27] in their work of reinforcement learning for game playing, utilizing the time dependency conveyed by a temporal sequence of states can improve the observability of an agent. The sequence of sensor data $\bar{o}^t$ contained in our robot state representation seamlessly matches this reported finding. Motivated by those authors' work, we propose to utilize the recurrent neural network to capture the time-dependent information within $\bar{o}^t$, and project $\bar{o}^t$ to low-dimensional embedding e^t, in order to deal with the high-dimensionality of state space. Hausknecht et al. [27] utilized a recurrent neural network to process a temporal sequence of image frames. In contrast to their mechanism, we are applying a recurrent neural network (i.e., LSTM) to robot navigation and use it to deal with high-dimensional range sensor data. Figure 4.2 illustrates an example of mapping a state of a robot to action through the designed recurrent-neural-network policy.

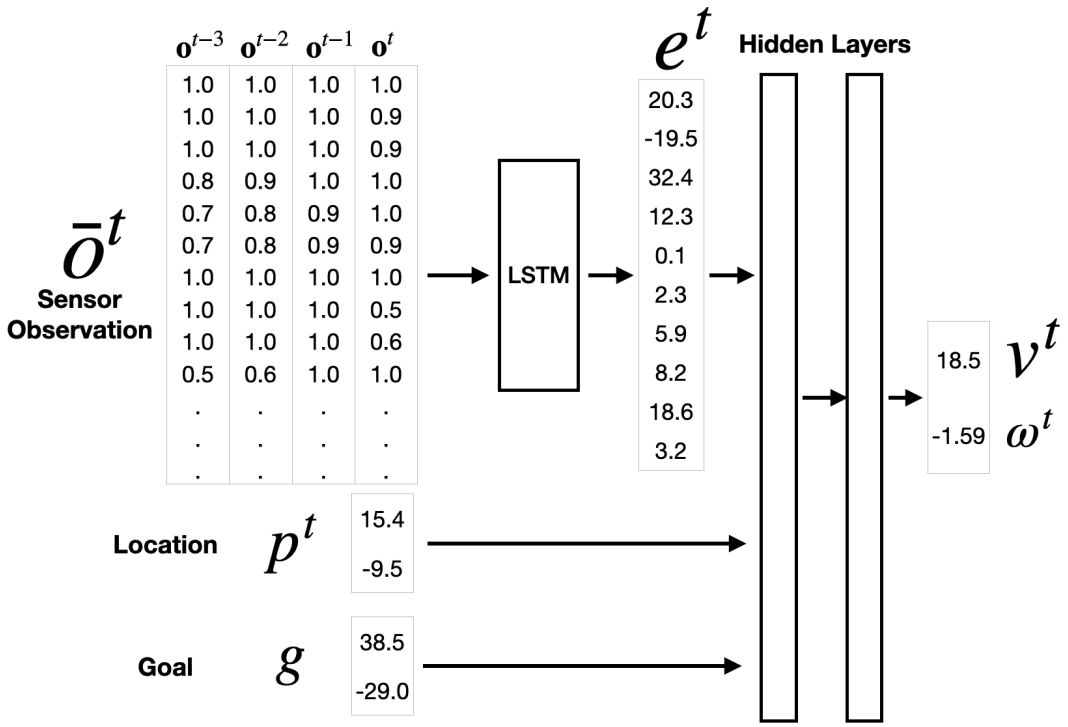


Figure 4.2: An example of mapping sensor data to actuation commands through a neural network policy.

4.5 MCA in Practice: Multi-Task & Multi-robot

The physical world poses a highly complex environment. For mapless collision avoidance (MCA), a robot may need to handle various situations or cooperate with other robots. As long-standing issues in reinforcement learning, multi-task, and multi-agent learning had been actively studied in the past. It will continue to bring benefits to deploying reinforcement learning to practice.

Any real-world task will probably be too complicated for a robot to solve through trials and errors directly. It is common to decompose a complex task into simpler sub-tasks and have a robot learn a multi-task policy to solve each individual of them. With a multi-task policy, one would expect a robot to have asymptotic optimal performance for anyone of the sub-task. That says, an effective multi-task policy should let a robot have an optimal and balanced performance on a collection of relevant tasks without salient performance on any specific ones. Besides, a multi-task policy should be

general and allows a robot to handle a more complex task which composes those sub-tasks.

In practice, many tasks involve multiple robots. Robots can cooperatively solve a task by learning their policies through multi-agent learning. Broadly, the multi-agent learning can be categorized as joint policy learning and individual policy learning. This thesis focuses on the later. Particularly, this thesis studies learning individual policies for robots to operate based on local sensor observation in a cooperative environment. It is well known that the interaction among robots can make an environment non-stationary and can make multi-agent learning diverge. To make an environment stationary, it is assumed that each robot follows a separate MDP. In addition, all the robots take action at the same time at each time step. Under this setting, an effective multi-agent learning method should enable robots to take coordinated actions based on their sensor observation using their own policies.

4.6 Quality Measurement

The objective of applying reinforcement learning to mapless collision avoidance is to learn an optimal policy for a robot to navigate in particular environments without collisions. In the context of mapless collision avoidance, the quality of a reinforcement learning method can be measured according to 1) its learning performance with respect to a certain reward function, 2) and the performance of its learned policy in terms of navigation behavior.

The reward function utilized by a reinforcement learning directly impacts the performance of the policy learned the method. It is common to define such a reward function based on criteria related to robot navigation performance measurement. An effective policy should enable a robot to navigate its initial location to its goal as

quickly as possible without collisions. Besides, for safe navigation, robots should avoid moving too close to the other robots.

Motivated by the above objective, we evaluate robot navigation according to the following criteria:

success rate: the probability that a robot can complete navigation;

time step: the number of time steps for a robot to complete navigation;

minimum distance: the closest distance between robots during navigation;

In this thesis, we study reinforcement learning for mapless collision avoidance in simulated environments, which is the pre-requisite step for deploying reinforcement learning in the physical world. The above criteria intend to describe the quality of robot navigation with a policy learned in a simulated environment. In addition to those criteria, we also evaluate the learned policy to a different number of robots within the same environment for multi-robot collision avoidance.

Chapter 5

Real-Time RL Architecture for Mapless Collision Avoidance

Applying reinforcement learning to real-world environments needs to deal with real-time operation of a robot [19]. Inspired by impressive results achieved in artificial domains, off-policy model-free reinforcement learning methods have been actively studied for navigation [51], collision avoidance [16], and grasping [63]. To be consistent with existing reinforcement learning literature, this chapter refers a robot as an agent.

In a typical reinforcement learning setting, an agent follows a Markov Decision Process (MDP) and plans its actions in discrete time. As reinforcement learning comes to a real-world environment, an agent must discretize continuous time into a certain resolution and take action in real-time. At each time step, a robot needs to process its sensor data, inferring an action, and actuate its action. In general, the action inferring step may also involve improving the robot's controller through a learning method. In real-time systems, control delay is ubiquitous and is well-known for destabilizing system performance [59]. Specifically, this thesis defines control delay as a time delay before a robot actuates an action at a particular time step. Figure 5.1 shows an example of control delay.

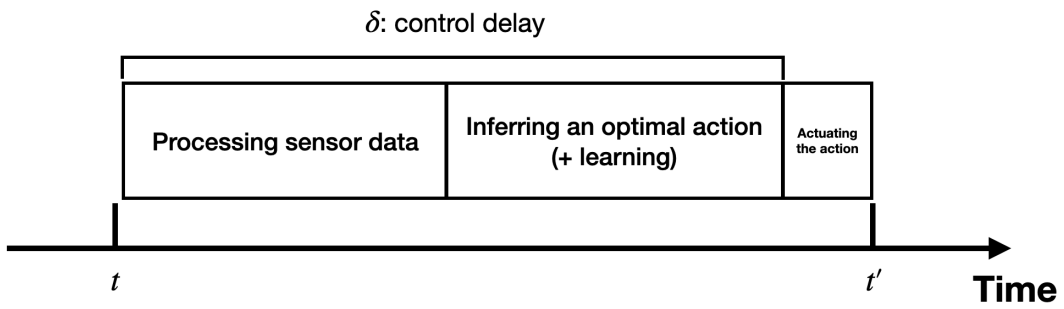


Figure 5.1: An example of control delay.

Because of the stochastic nature of an MDP, control delay can be adapted by a transition function. However, an environment can become Non-Markovian, if control delay has high variance. Addressing this issue, we propose a real-time actor-critic architecture (RTAC) for applying off-policy reinforcement learning methods for mapless collision avoidance. RTAC stabilizes control delay by decoupling policy learning from environment interaction through multiple threads. Besides, it has the scalability of asynchronous reinforcement learning. RTAC is evaluated in a series of environments simulated close to real-world settings. Those environments are physics-enabled, where agents learn to map high-dimensional sensor data to continuous actions. In extensive experiments, we demonstrated the effectiveness of RTAC in terms of stabilizing control delay, improving learning performance, and supporting asynchronous reinforcement learning.

Because learning optimal policy requires a large number of trials and errors, a practical application of reinforcement learning usually involves learning policy in simulation and deploying the learned policy to a real-world environment in the second phase. It is crucial to reduce the *reality* gap [91] and incorporate control delay in simulation to enable successful policy transfer. Schuitema et al. [73] extended the Q-learning and SARSA algorithms to incorporate delay time and archived improved performance in simulation. Hester et al. [30] proposed a real-time architecture for applying model-based reinforcement learning to autonomous vehicle control. Unlike the

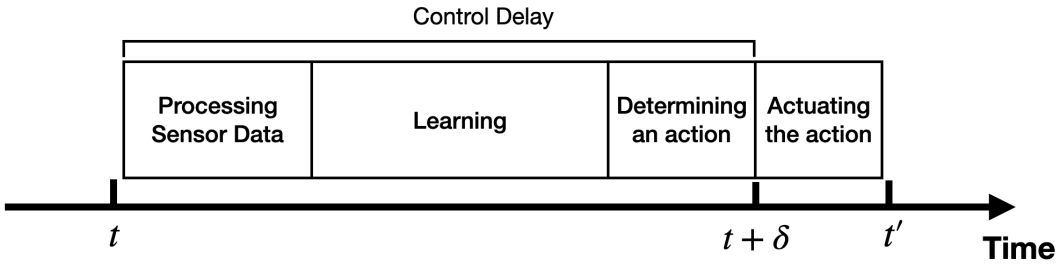


Figure 5.2: Control delay in reinforcement learning.

previous work, RTAC is based on the recent advancement in model-free off-policy reinforcement learning [50, 79, 96], and uses experience replay [1] to improve the sample efficiency of a learning method. Besides, it well supports asynchronous reinforcement learning even in the presence of control delay.

5.1 MDP under Control Delay

Reinforcement learning models the interactions between an agent and an environment as a Markov Decision Process (MDP). An MDP is a tuple (S, A, T, R), where S and A are the state and action spaces of an agent. $T : S \times A \times S \to \mathbb{R}$ is a state transition function, and it defines a probability distribution over the states resulting from an action execution. $R : S \times A \times S \to \mathbb{R}$ is a reward function, and it determines a reward of an agent-based on the consequence of taking action in a specific state. In MDP, an agent takes actions in discrete-time, and incrementally constitutes a state-action trajectory. By definition, an environment is Markovian, if the probability distribution over the future states that an agent will visit only depends on the present state of the agent.

It is crucial to ensure the Markovian property of an environment in order to apply reinforcement learning to a control task. However, in practice, an agent operates in continuous time and always has time delay before actuating an action determined in an observed state [59]. The time delay δ for an agent to take action is defined

as control delay. Figure 5.2 illustrates an example of control delay in the context of reinforcement learning. In the example, the learning step's performance depends on an underlying policy optimization method, and a robot can take an arbitrary amount of time to complete this step. In this case, this learning step can cause control delay δ to have high variance, particularly when the robot has limited computing resources.

Let s_t be the state observed by an agent at time t, and the agent determines its optimal action based on s_t. When δ has high variance, s_t can be arbitrarily different from the state $s_{t+\delta}$ where the agent actuate the action. In this case, the optimality of the determined action can not be guaranteed. To ensure the optimality of the determined action, the agent needs to predict $s_{t+\delta}$ based on s_t and determines its action based on the predicted state. In this case, one shall consider $s_{t+\delta}$ to be the present state, and determining an action involves the historical state s_t. This violates the Markovian property of an environment. Therefore, δ with high variance is prohibitive in reinforcement learning applications.

A MDP is able to tolerate control delay δ with low variance, as long as δ meets the real-time constraint of an agent (i.e., $\delta < t' - t$). Because of the stochastic nature of an MDP, the transition function T can incorporate the effect of the δ into the probability distribution over the transited states. Although the magnitude of δ does not affect Markovian property, it poses challenges for deploying reinforcement learning policy in practice, particularly for safety-critical tasks.

5.2 RTAC

The high variance of control delay can destabilize reinforcement learning and can cause an environment being Non-Markovian. As illustrated in Figure 5.2, the step of learning a robot policy can take arbitrary amount of time and causes high variance of control delay for the robot. To reduce the variance of control delay, the policy learning

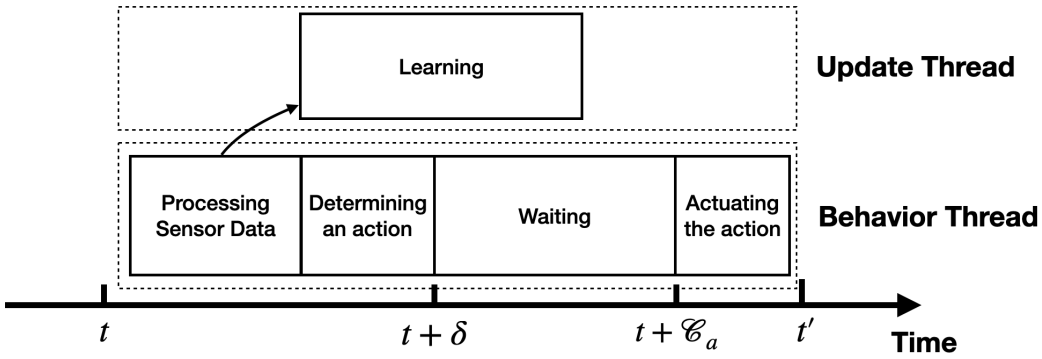


Figure 5.3: Decoupling environment exploration and policy learning of a robot through two threads.

is decoupled from environment interaction of the robot by two threads, i.e.; the update thread and the behavior thread. Figure 5.3 shows such decoupling mechanism.

In the decoupling mechanism, the update thread focuses on optimizing a robot's policy based on the robot's experience. The behavior thread focuses on determining actions based on the sensor data perceived by the robot. In this case, the control delay of the robot only depends on the behavior thread's performance. As the policy of the robot is deterministic (as the policy shown in Chapter 4), the behavior thread of the robot can use the policy to map perceived sensor data to a particular action quickly. Therefore, determining an action with the behavior thread only introduces a small control delay. To further reduce the variance of control delay, it is desired to make the robot to have constant control delay in any time step. To this end, the concept of *estimated control delay for actuation* $\mathscr{C}_a$ is introduced. The $\mathscr{C}_a$ is empirically determined, and it represents the upper bound of control delay that the robot will have in any time steps. At each time step, the behavior thread of the robot is expected to wait until its control delay being equal to $\mathscr{C}_a$, before actuating a determined action. In this way, one can expect that, at any time step, the control delay of the robot is always equal to $\mathscr{C}_a$. It is important to note that $\mathscr{C}_a$ should be reasonably small so that the robot can meet its minimum control frequency.

Based on the above decoupling mechanism, RTAC is proposed for applying model-free off-policy reinforcement learning to real-time control. Figure 5.4 shows the proposed architecture.

5.2.1 Stabilizing Control Delay

The proposed work assumes that an agent represents its update policy and behavior policy as parameterized functions (e.g., neural networks). Also, the update policy and the behavior policy share the same set of parameter variables. RTAC stores the values of the parameter variables either in a shared memory or in parameter servers. During learning, the behavior thread periodically (i.e., every t time steps) updates the behavior policy by copying the shared parameter values. On the other hand, the update thread uses DDPG [50] algorithm to evaluate policy gradients and apply them to the shared parameter values, and evolves the update policy by copying the updated shared parameter values to its local memory. Those copying operations can be greatly accelerated by GPUs and can have very low latency.

In RTAC, each pair of update and behavior threads jointly maintains a replay buffer. Although the synchronization between those two threads can introduce some overhead, such overhead can be accommodated by the estimated control delay for actuation $\mathscr{C}_a$. In an actor-critic method, an agent alternates between improving its Q-function and optimizing its policy. In RTAC, the Q-function is completely local to the update thread, and the update thread optimizes its Q-function and update policy in a sequence.

Without decoupling policy learning from environment exploration, an agent typically adopts a sequential architecture, where it improves its update policy every certain time steps or after certain trajectories are collected [57, 75]. With the sequential architecture, policy optimization can hinder environment interaction, since an agent will not determine its action until policy optimization is complete. If the policy

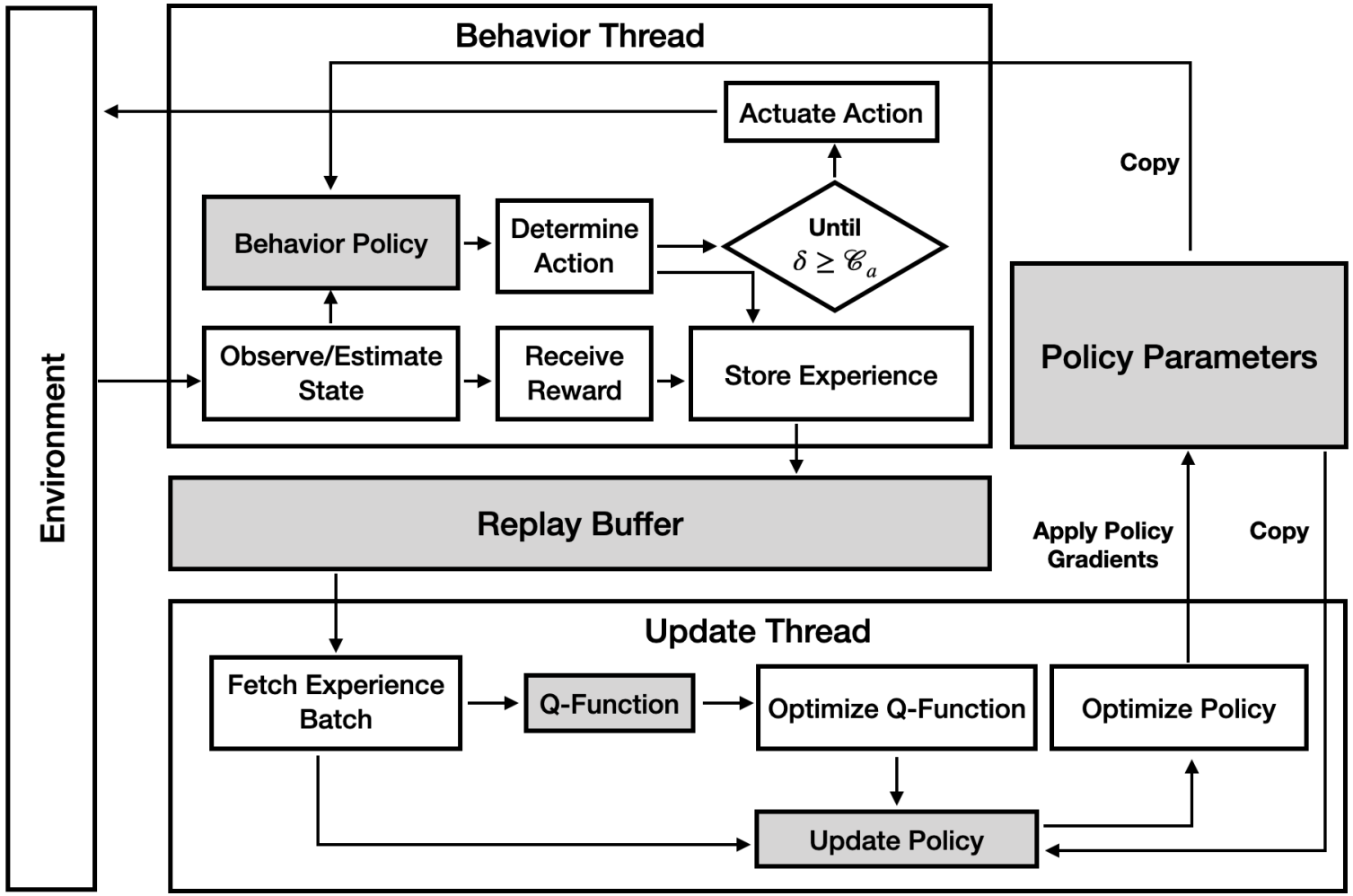


Figure 5.4: Real-time actor-critic architecture (RTAC).

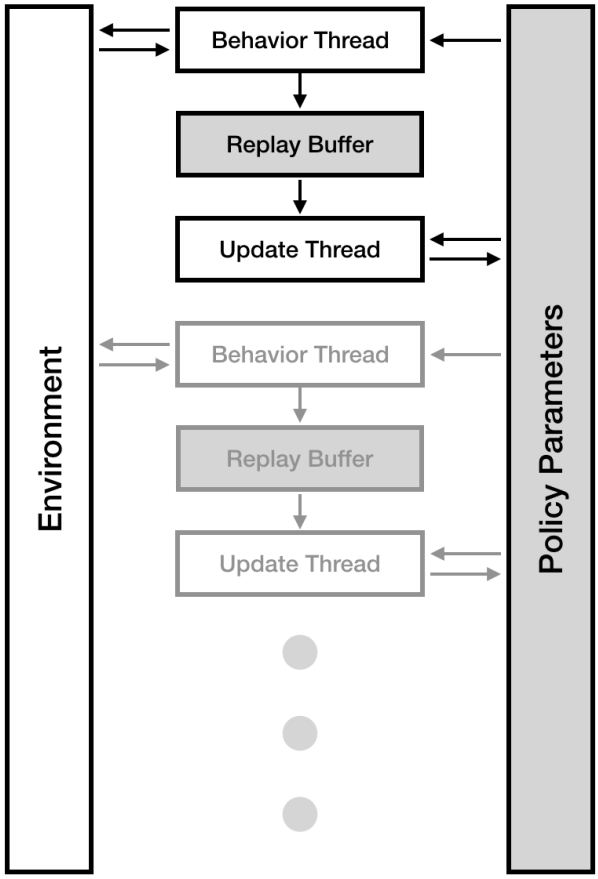


Figure 5.5: The overview of the asynchronous RTAC

optimization happened before the end of an episode, it introduces additional time delay and increases the variance of control delay. Although an agent could optimize the update policy at the end of an episode, the sequential architecture can slow down the learning process, particularly when the optimization is computationally demanding (e.g., computing returns of roll-out trajectories).

5.2.2 Scalability of Asynchronous Learning

A practical application of reinforcement learning typically involves training policy in simulation and transferring the learned policy to a real-world environment in the following phase. As learning an effective policy requires a large amount of trials and errors, a parallel training mechanism is desired and adopted by recent reinforcement

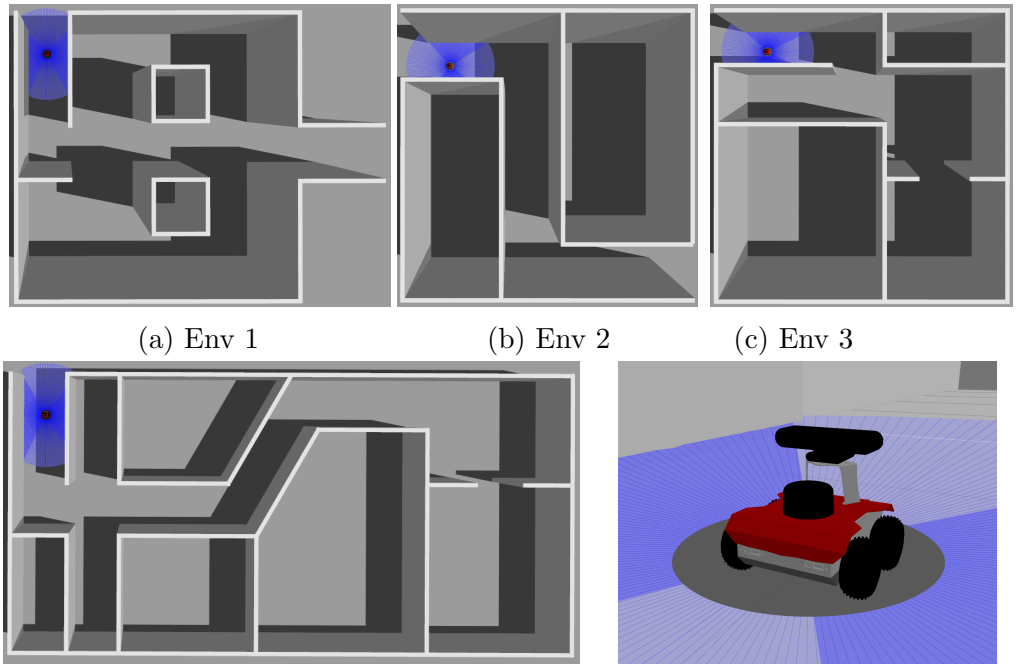
learning methods [55, 38]. To support asynchronous reinforcement learning, RTAC allows multiple worker agents to simultaneously learn in a different environment and synthesize the knowledge they have learned into a shared policy. Figure 5.5 presents the asynchronous RTAC.

In an asynchronous RTAC, each agent has its own update thread, behavior thread, and replay buffer. During learning, each agent situates in its own environment and accumulate its experience in its own replay buffer through its behavior thread. All the worker agents share the same set of policy parameter variables, and the update thread of each agent applies its policy gradients to the shared policy parameter values either synchronously or asynchronously. Such a policy optimization mechanism enables various off-the-shelf asynchronous reinforcement learning methods [55] to be applicable to real-time systems.

It is important to note that the asynchronous RTAC requires worker agents to use the same $\mathscr{C}_a$ in their update threads. Otherwise, the experience collected by different agents can be inconsistent and hinders the convergence of the share policy parameters. In the asynchronous setting, RTAC uses separated replay buffers for worker agents. This reduces the overhead caused by merging the experience collected in different environments.

5.3 Evaluation

This section demonstrates that the RTAC architecture, which utilizes the proposed recurrent neural network policy, can address the following issues that may arise in real-world reinforcement learning applications, including partial observability, high-dimensional sensor data, and real-time control delay. This section focuses on evaluating the RTAC architecture, in terms of dealing with real-time control delay during



(a) Env 1 (b) Env 2 (c) Env 3

(d) Env 4 (e) Agent

Figure 5.6: The agent and environments used in experiments

training and learning effective policies that are represented as deep recurrent neural networks.

The evaluation is based on the well-known reinforcement learning algorithm, DDPG [50]. By comparing the performance of DDPG with and without RTAC, experiments demonstrate the effectiveness of RTAC, in terms of learning effective recurrent neural network policies with a single agent under real-time constraint. To show that RTAC can support asynchronous reinforcement learning, RTAC is used as the execution architecture of the Asynchronous DDPG [103]. The Asynchronous DDPG is the DDPG algorithm that are executed with multiple threads, where each thread uses DDPG to generate gradients and asynchronously apply the gradients to a shared policy. In the context of RTAC, such a thread is a worker agent.

5.3.1 Simulation Details

The evaluation of RTAC is based on navigation tasks in simulated maze-like environments. The proposed task environments are physic-enabled and implemented using ROS [65] and Gazebo [41]. Figure 7.3c to 5.6d illustrates those environments. In each environment, an agent needs to plan a collision-free path from a fixed initial location to a pre-defined goal in real-time. An agent is simulated as a mobile robot, i.e., a ROSbot. The robot has a simulated LiDar sensor, which has a range of 0.1m to 2m and can scan its surrounding environment with 180 evenly spaced lasers. During navigation, an agent moves at a constant speed and controls its moving direction through rotational velocities. Table 5.1 summarizes the hyper-parameters used for policy learning.

5.3.2 Relate Simulation to Theoretical Formulation

In this chapter, the evaluation of proposed architecture, RTAC, is based on the DDPG algorithm [50]. During training, DDPG allows an agent to alternate between learning its policy and q-function through stochastic gradient optimization. In this chapter, we represent a policy and q-function as deep neural networks. The structure of the policy is presented in Figure 4.2. The q-function has almost identical neural network

hidden layer neurons	1024
number of hidden layers	2
LSTM hidden state size	128
hidden layer activation	ReLu
actor learning rate	0.00001
critic learning rate	0.001
target network τ	0.01
batch size	256
relay buffer size	100000

Table 5.1: Hyper-parameters used in experiments for learning policies.

architecture, except for that its take an action (i.e., a pair of scalar values) as addition part of the input and outputs a single scalar value as the q-value of the input action.

Based on the above neural network representation, DDPG allows an agent to compute the gradients for optimizing its policy and q-function according the following equation 5.1 and 5.2

$$.\nabla_\theta J(\mu_\theta) = \mathbb{E}\{\nabla_\theta \log \mu_\theta(s_t) \nabla_{a_t} Q(s_t, a_t) \mid_{a_t = \mu_\theta(s_t)}\} \tag{5.1}$$

$$\nabla_w L(w) = \mathbb{E}[\nabla_w (r + \gamma Q(s_{t+1}, a_{t+1}; w) - Q(s_t, a_t; w))] \tag{5.2}$$

In both equations, s_t and a_t are the state and action of an agent at time t. r_t is the reward received by the agent after taking a_t in s_t. We θ and w to denote the weights of the neural networks representing the agent's policy and q-function respectively. In simulation, s_t, a_t and r_t have the definition given in Chapter 4, and those definition is all based on the data that the agent can perceive in the Gazebo simulation. At each time step t, the Gazebo simulator will send those data to the training program which executes the DDPG algorithm. Upon receiving those data, the training program will structure those data as the state, action and reward of the agent. Then, it stores those state, action and reward into a buffer. As the agent repeatedly navigate in the Gazebo simulation, the training program fills the buffer with the transactions, (s_t, a_t, r_t), based on the data received at different time steps.

To compute the gradients based on the above equations, the training program first samples multiple transactions, $(s_t, a_t, r_t, s_{t+1}, a_{t+1})$, from the buffer. For each sampled transaction, the training program computes gradients for θ and w based on those two equations without expected value notation $\mathbb{E}$. To compute the expected gradient values, the training program will then average the gradients values that were computed based on those transactions for each of the parameters contained in θ and w respectively. When multiple agents are navigating in the Gazebo simulation, the

training program will store the state, action and reward of those agents in separate buffers. Then, the training program can compute expected gradient values according to the above procedure for each of those simulated agents.

5.3.3 DDPG with RTAC

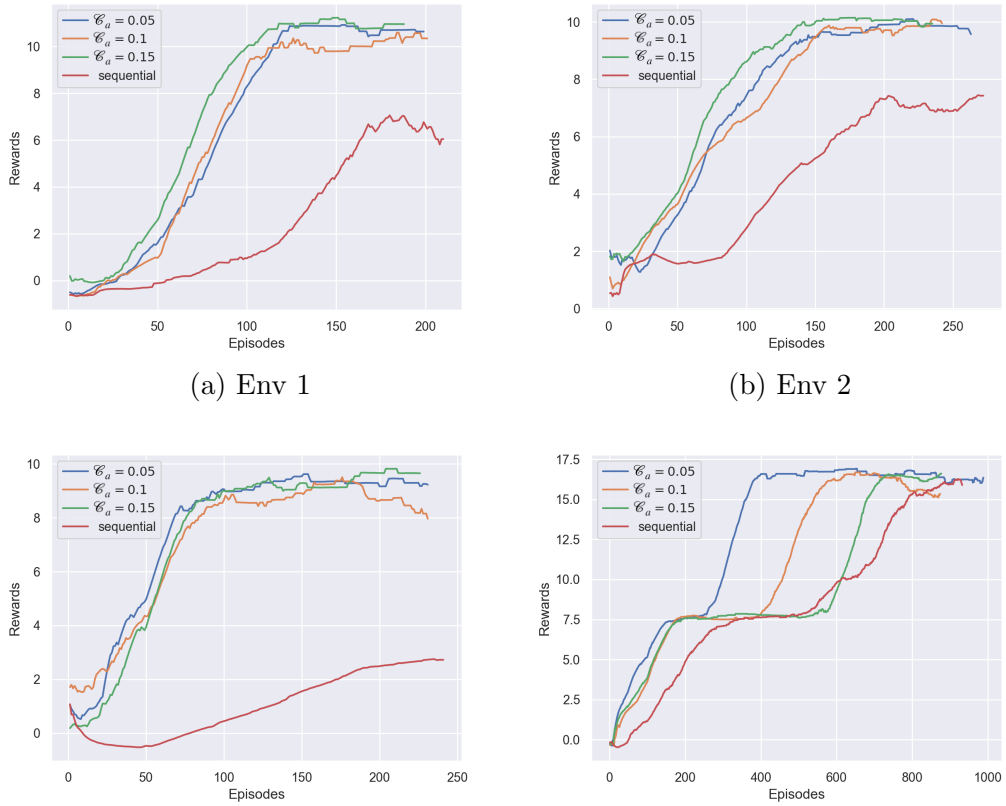
RTAC reduces the variance of control delay by decoupling policy learning and environment interaction in order to stabilize an actor-critic method applied to a real-time system. In contrast, when an agent conducts policy learning and environment interaction in sequence, the time delay for optimizing the policy can increase the variance of control delay, particularly in systems with limited computing resources, such as robots. Table 5.2 compares the RTAC with the sequential architecture, in terms of their control delay statistics.

	$\mathscr{C}_a = 0.05$		$\mathscr{C}_a t = 0.1$		$\mathscr{C}_a t = 0.15$		sequential	
	var	mean	var	mean	var	mean	var	mean
Env 1	1.925e-05	0.051	2.431e-05	0.101	1.466e-05	0.151	0.004	0.127
Env 2	1.399e-05	0.051	3.739e-05	0.101	5.582e-06	0.151	0.003	0.123
Env 3	2.557e-05	0.051	1.680e-05	0.101	2.041e-05	0.151	0.004	0.129
Env 4	4.055e-06	0.051	1.196e-05	0.101	1.210e-05	0.151	0.004	0.141

Table 5.2: Control delay statistics for RTAC and the sequential architecture.

The results presented in Table 5.2 is based on the average of mean and variance of control delay in 50 episodes. In the experiments, compare the control delay in RTAC based on different estimated control delay for actuation $\mathscr{C}_a$. As $\mathscr{C}_a$ is increased, the variance of control delay does not change significantly across all environments. In contrast, the sequential architecture causes much higher variance.

The performance of DDPG with the RTAC and the sequential architecture is compared, in terms of a moving average of episodic rewards. Figure 5.7 presents the results collected in all environments. According to the results, the DDPG with RTAC significantly outperforms the DDPG with the sequential architecture. Accord-



(a) Env 1 (b) Env 2

(c) Env 3 (d) Env 4

Figure 5.7: The moving averages of episodic rewards achieved through RTAC under various control delay. Video demo: `https://youtu.be/6dH7-0Miu7c`

ing Table 5.2, the sequential architecture causes much higher variance of control delay, although the mean is in the reasonable range (i.e. $[0.05, 0.15]$ in our experiments). With different estimated control delay for actuation $\mathscr{C}_a$, DDPG with RTAC achieved similar learning performance. This proves our claim that the high-variance of control delay can make an environment Non-Markovian, but the magnitude of control delay does not effect the Markovian property, as long as it is in a reasonable range. Table 5.3 summarize the performance of the policy learned by DDPG under RTAC architecture.

According the results, the RTAC with different $\mathscr{C}_a$ can consistently enable DDPG to achieve good success rate, compared to the sequential architecture. When we the $\mathscr{C}_a$ to be a smaller value, the agent can take action with frequency. Therefore, the time

	$\mathscr{C}_a = 0.05$		$\mathscr{C}_a t = 0.1$		$\mathscr{C}_a t = 0.15$		sequential	
	succ	timestep	succ	timestep	succ	timestep	succ	timestep
env1	0.93	73.20	0.86	49.52	0.96	49.22	0.22	51.34
env2	0.86	70.06	0.89	48.50	0.90	48.64	0.02	48.36
env3	0.918	60.56	0.69	62.54	0.97	41.26	0.00	52.18
env4	0.88	120.92	0.78	103.30	0.97	92.50	0.84	85.10

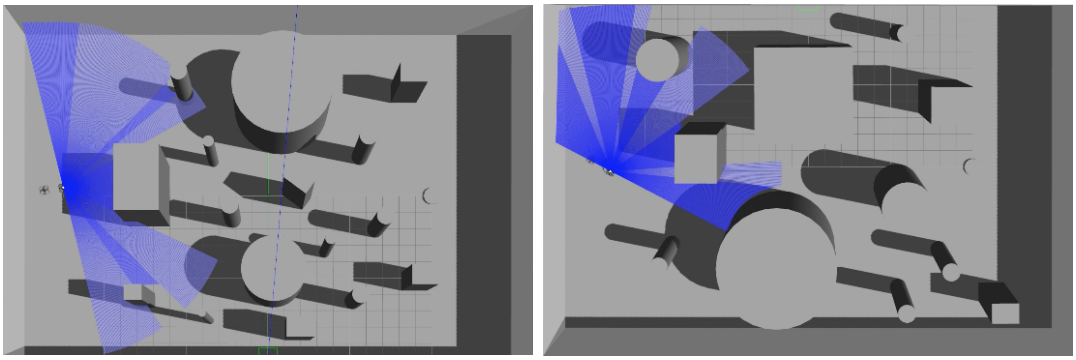
Table 5.3: The performance of the policy learned through both the RTAC and the sequential architectures. The metrics summarized in the table includes, the success rates and the time steps for an agent to complete navigation.

steps for an agent to complete navigate increases, as we decrease the $\mathscr{C}_a$. Notice that the DDPG with sequential architecture achieved good success rate in env4. In our experiments, we found that the control delay does not have significant impact on the control of an agent, when the agent is moving at constant speed towards a particular direction. As shown in Figure 5.6d, the hallway-like regions requires an agent to move in the constant speed toward a particular direction. This gives opportunities to the DDPG with the sequential architecture to learn an effective policy. In general, the RTAC still outperformed the sequential architecture and enabled the DDPG to learn better optimal policies in the most of the environments.

With the RTAC architecture, we used DDPG to learn policies in clustered environment where obstacles are randomly distributed. In this part of the experiments, we set the $\mathscr{C}_a$ to be 0.1 seconds. Figure 5.8 shows those environments, and Table 5.4 summarizes the performance of the policies learned in those environments. Those results demonstrate that RTAC can also enable DDPG to learn effective policies under different random obstacle distributions. A video demonstration can be accessed through `https://youtu.be/or6zQ6z4fqs`.

	$\mathscr{C}_a = 0.1$	
	succ	timestep
env5	0.84	119.84
env6	0.87	128.38

Table 5.4: The performance of the policies learning Env5 and Env6. The $\mathscr{C}_a$ is set to be 0.1 during training.



(a) Env5 (b) Env6

Figure 5.8: The environments with random obstacle distribution. We used Ardrone agent model to learn collision avoidance policy through DDPG based on the RTAC architecture. A video demonstration can be access through `https://youtu.be/or6zQ6z4fqs`

.

5.3.4 Asynchronous Reinforcement Learning

RTAC is applicable to an asynchronous setting. To learn a policy, asynchronous RTAC consists of multiple pairs of behavior and update threads, and each pair of threads corresponds to a worker agent. Those worker agents can optimize globally shared policy parameters in parallel. This part of the experiment conducts the asynchronous reinforcement learning using a single PC, and implement the shared parameter variables as shared memory. In the asynchronous setting, each agent learns its environment through DDPG and applies its policy gradients asynchronously to the shared policy parameters.

For the asynchronous reinforcement learning, RTAC is evaluated based on the environment presented in Figure 5.6d, which consists of all the patterns presented in Env 1 to Env 3. Since the asynchronous RTAC has multiple worker agents learn the shared policy in parallel, one can expect the learned policy to represent more complex behavior, comparing to the policy learned by a single agent. Figure 5.9 shows the performance of the asynchronous reinforcement learning under various delay thresh-

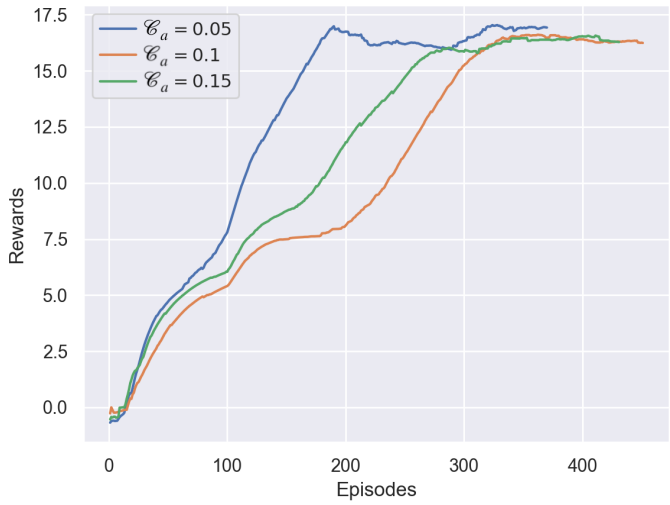
olds. Table 5.5 summarizes the control delay during the asynchronous reinforcement learning.

As shown on the right side of Figure 5.9b, three worker agents are used to learn the same environment, and each worker agent interacts with its own copy of the environment. For evaluating the policy jointly optimized by those three worker agents, every 5 seconds, a separate evaluation agent makes a copy of the jointly optimized policy and evaluates its performance in terms of episodic rewards. The left side of Figure 5.9b shows the evaluation agent.

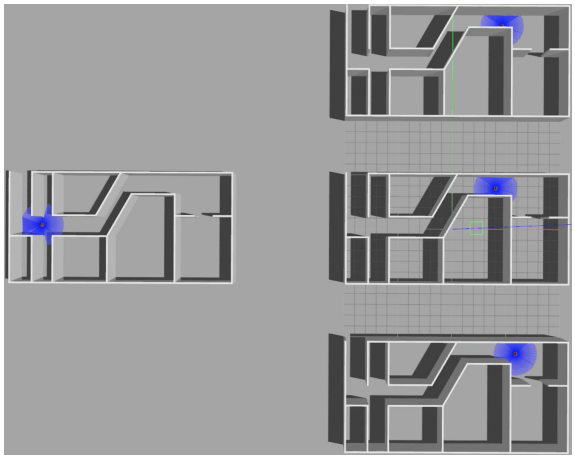
	$\mathscr{C}_a = 0.05$		$\mathscr{C}_a t = 0.1$		$\mathscr{C}_a t = 0.15$	
	var	mean	var	mean	var	mean
Agent 1	1.559e-04	0.052	5.642e-05	0.102	2.584e-05	0.152
Agent 2	1.390e-04	0.052	3.148e-05	0.102	2.948e-05	0.152
Agent 3	1.591e-04	0.053	6.194e-05	0.102	4.441e-05	0.152

Table 5.5: Control delay statistics for asynchronous RTAC.

According to the results, RTAC enables the asynchronous DDPG to converge at optimal policies under all three delay thresholds (i.e., 0.05, 0.1, 0.15). Each experiment initializes the shared policy with random parameters, and the difference in terms of convergence time is caused by those initial parameter settings, instead of the estimated control delay for actuation $\mathscr{C}_a$. According to Table 5.5, the asynchronous RTAC does not significantly increase the variance of control delay, and it can ensure each worker environment being Markovian throughout the learning. Besides, the results also prove that an MDP could tolerate stabilized control delay even in an asynchronous setting. Table 5.6 summarizes the performance of the policy learned under the asynchronous setting.



(a) The episodic reward moving average.



(b) Environments for asynchronous reinforcement learning.

Figure 5.9: Asynchronous reinforcement learning with RTAC. Video demo: `https://youtu.be/6dH7-0Miu7c`

5.4 Summary

This chapter presents a reinforcement learning architecture RTAC, which applies an actor-critic method to mapless collision avoidance. In this chapter, we showed that the control delay with high variance can destabilize the learning performance of an agent operating in real time. We evaluated the RTAC based on navigation tasks in simulation. In those navigation tasks, the simulated agents operate in real time and

	$\mathscr{C}_a = 0.05$	$\mathscr{C}_a = 0.1$	$\mathscr{C}_a = 0.15$
succ	0.816	0.91	0.979
timestep	125.81	108.63	91.59

Table 5.6: The performance of the policy is learned under asynchronous RTAC. The metrics summarized in the table include the success rates and the time steps for an agent to complete navigation.

navigate in physics-enabled environments. In extensive experiments, we demonstrated that RTAC is salable and allowed multiple worker agents to asynchronously optimize a shared policy under real-time constraint.

RTAC used DDPG as the learning algorithm for optimizing policies of agents. It is straight forward to incorporate alternative off-policy actor-critic methods into RTAC for learning either deterministic or stochastic policies. Besides, one can also combine RTAC with more advanced experience replay techniques, such as prioritized experience replay [72] and hindsight experience replay [5]. Although a small number of agents is used to evaluate the asynchronous RTAC, the overall task to be learned is still challenging. Using a smaller number of worker agents allows us to better focus on the effects of control delay on asynchronous reinforcement learning in a close-to-real-world setting. Specifically, during the asynchronous reinforcement learning, each worker agent learns to map high-dimensional sensor data (i.e., 720 dimensions) to continuous actions and asynchronously update over 1 million parameters of the shared policy.

The proposed RTAC is for the off-policy model-free reinforcement learning. Although RTAC is based on the recent advancement in reinforcement learning, in practice, many control tasks still favor model-based methods, particularly for robotic tasks. The future work would expand the study presented in this chapter to model-based methods. It would be attractive to have a unified architecture for applying either model-based or model-free methods to real-time systems.

Chapter 6

Multi-Task Mapless Collision Avoidance

A robot navigating in the physical world may need to avoid collisions in different scenarios. We define a scenario as an environment with a known obstacle distribution. In this chapter, we consider mapless collision avoidance in a scenario as a particular task. In this case, the physical world is a multi-task environment for robot navigation. Manually designing a controller for mapless collision avoidance in a multi-task environment require extensive parameter turning and engineering effort. It is attractive to have a robot learn a multi-task policy for mapless collision avoidance through reinforcement learning. In the rest of the chapter, we refer a robot as an agent, in order to be consistent with existing reinforcement learning literature.

Deep reinforcement learning was able to exceed human performance in various control tasks [78, 56, 50]. Even though the results are impressive, the existing work normally trains a specialized policy from scratch for one task at a time, and each task requires training a different policy instance. Learning a specialized policy for a task in a complex environment (e.g., the physical world) requires a tremendous amount of time and agent experience, making the deep reinforcement learning methods sample

inefficient. To address this issue, researchers in the reinforcement learning community shifted their attention to multitask reinforcement learning, which could estimate a general policy by learning a series of related tasks, either sequentially or simultaneously. Compared to single-task reinforcement learning, one would expect that in multitask reinforcement learning, learning each task requires much fewer data. Combining solutions to multiple tasks enables a policy to have better asymptotic performance and generalizability.

Learning simple tasks individually does not make the learning in a multitask setting simpler. Instead, it poses at least two stressing issues for learning effective policies. First, the processes of learning individual tasks often interfere with each other. When policy parameters are jointly optimized based on multiple tasks, the gradients evaluated in one task can likely override the gradients evaluated in another task. Without special treatments, this would make a multitask reinforcement learning method is sample inefficient. Second, a multitask policy can have an unbalanced performance on learned tasks. Since tasks can be learned based on rewards with different scale or distribution, some of the tasks can be more *salient* than the other during training [29]. Addressing those issues, recently parallel multitask reinforcement learning [20, 29] has demonstrated remarkable effectiveness in Atari games [56] and DeepMind Lab [9]. In those environments, agents operate in discrete action spaces. On the contrary, multitask reinforcement learning for continuous control is still underexplored. Here, the chapter refer to robots as agents so that the proposed work can be better related to existing reinforcement learning literature.

Agents with continuous actions are commonly involved in robotic control tasks, such as autonomous driving [71], UAV control [14] and object manipulation [66, 35]. Unlike game environments [56, 9], robotic control is often constrained on physical factors, e.g., limited sensing ranges, high-dimensional sensor data, and limited acceleration of motion. In this chapter, we study multitask reinforcement learning for

mapless collision avoidance. In particular, we focus on agents with continuous actions and partial observability. Based on Deep Deterministic Policy Gradient (DDPG) algorithm [50], we presents an asynchronous method, Asyn^{d}-DDPG, for learning a shared policy and q-function with multiple simultaneous worker agents. To deal with partial observability of agents, Asyn^{d}-DDPG represents the shared policy and q-function as recurrent neural networks that allow agents to take actions based on a sequence of recent sensor observation.

We empirically found that ensuring the sparsity of the gradients applied to the shared policy and q-function can reduce conflicts in learning competing tasks and avoid unbalanced learning. To this end, the proposed work regularizes the shared policy and q-function using Dropout to ensure the gradients generated through back-propagation are sparse. As each agent needs to evaluate its sparse gradients, applying different Dropout to the same neural network requires synchronization. To solve this issue, Asyn^{d}-DDPG let each worker agent maintain up-to-date copies of the shared policy and q-function in its memory and independently apply Dropout regularization to those local copies. During training, each worker agent asynchronously updates the shared policy and q-function using the gradients evaluated based on its regularized local policy and q-function.

Asyn^{d}-DDPG is evaluated in physic-enabled environments based on robotic simulation. In experiments, worker agents simultaneously learn different navigation tasks in a small number of maze-like continuous environments. Experimenting with those environments provides the first step to understanding how Dropout regulation affects agents' learning performance in a multitask setting. It also allows us to analyze the performance of the policy learned by Asyn^{d}-DDPG on all those learned tasks in detail. In experiments, we demonstrate that Dropout regularization can effectively reduce the interference among competing tasks and enable a learned policy to have a balanced performance on individual tasks. With extensive evaluation, the policy

learned by Asyn^{d}-DDPG can significantly outperform the specific policies learned by DDPG in all test environments. Also, the policy learned by Asyn^{d}-DDPG can avoid collisions in more complex navigation tasks that are unseen by agents during training.

6.1 Sparse Gradients through Dropout for Asynchronous RL

Asynchronous reinforcement learning are actively studied in recent literature. Asynchronous reinforcement learning attempts to optimize a policy through scholastic optimization with different threads. Here, each thread is called a worker agent. As each worker agent computes gradients independently, the gradients applied by worker agents can improved the policy parameters toward different directions, which can result in conflicts. As proven in [68], asynchronous stochastic optimization method can always converge on optimal parameter values, if the applied gradients are sparse. This section presents the Dropout regularization technique utilized by Asyn^{d}-DDPG to compute sparse policy gradients for asynchronous reinforcement learning.

As a reinforcement learning approach directly optimizing policies of the agent, policy gradient methods optimize a parameterized policy with respect to its expected reward using gradient decent algorithms. Let S define a state space of an agent; A be a set of actions the agent can take in each state $s \in S$. μ_θ is a policy with a parameter vector θ. When μ_θ is stochastic, the parameter gradients for optimizing its expected reward can be calculated as

$$\nabla_\theta J(\mu_\theta) = \mathbb{E}_{s\sim\rho^{\mu_\theta}, a\sim\mu_\theta}\{\nabla_\theta \log \mu_\theta(a \mid s) Q(s,a)\} \tag{6.1}$$

where ρ^{μ_θ} denotes the probability distribution that an agent visits each $s \in S$ using μ_θ. As a special case of Equation 6.1, Equation 7.1 computes the gradients for optimizing

a deterministic policy [79].

$$\nabla_\theta J(\mu_\theta) = \mathbb{E}_{s\sim\rho^{\mu_\theta}}\{\nabla_\theta \log \mu_\theta(s) \nabla_a Q(s,a) \mid_{a=\mu_\theta(s)}\} \tag{6.2}$$

$\nabla_\theta \log(\mu_\theta(s))$ is a Jacobian matrix where an entry in row i and column j represents the gradient of the ith parameter for the jth action. $\nabla_a Q(s,a)$ is a vector of gradients with respect to the Q-function for the action selected by μ_θ in a state. This chapter uses Q-functions to estimate policy gradients, and alternative estimations of policy gradients can be found in [74].

If a Q-function (or a value function) is unknown, an agent must fit the unknown Q-function based on its state-action trajectories collected online while calculating policy gradients for optimizing its policy. A method alternating between fitting a Q-function and optimizing a policy is called an actor-critic method.

6.1.1 Policy as Feed-Forward Neural Network

Suppose a policy μ_θ is represented as a feed-forward neural network with parameters θ. Let $L = \{1, 2, \ldots, l\}$ be the indexes of hidden layers. z^l and y^l are the input and output of the hidden layer l. In forward operation, dropout is applied to each y^l, s.t.

$$\tilde{y}^l = r^l * y^l$$

r^l is a vector whose components are independently sampled from Bernoulli distribution with a probability of P being 0, and P is called a dropout rate. $*$ denotes element-wise multiplication. $\tilde{y}^l$ is input to the hidden layer $l+1$ according to the following equation:

$$z^{l+1} = W^{l+1}\tilde{y}^l + b^{l+1}$$

$$y^{l+1} = \tau(z^{l+1})$$

where, $\tilde{y}^l$ is a column vector, and τ is non-linear activation function. W^{l+1} is a $n \times m$ weight matrix, where n and m are the number of neurons in hidden layers $l+1$ and l. For computing z^{l+1}, it's equivalent to zero out ith column of W^{l+1}, when ith component of r^l is 0. When the dropout rate P is sufficiently large, all weight matrices in a forward neural network can be sparse. While evaluating gradients for the weight matrices through back-propagation, the entries which were zeroed out in the forward operation are restricted to have gradients of 0. Therefore, the resulting gradient vector is sparse.

6.1.2 Policy as Recurrent Neural Network.

Taking Long Short Term Memory (LSTM) as an example, Dropout can be applied to a recurrent neural network to produce sparse gradients. Let x_t and h_t be the input and the output of LSTM at time t. Dropout is applied to LSTM in the following way

$$\tilde{h}_t = r^h * h_t$$

$$\tilde{x}_t = r^x * x_t$$

r^h and r^x are dropout vectors as what was explained in the feed-forward neural network case. t indicates the time step of a input sequence. With Dropout, LSTM is given by the equations below:

$$\begin{aligned}
\underline{i} &= sigm(U_i\tilde{h}_{t-1} + W_i\tilde{x}_t) & \underline{f} &= sigm(U_f\tilde{h}_{t-1} + W_f\tilde{x}_t) \\
\underline{o} &= sigm(U_o\tilde{h}_{t-1} + W_o\tilde{x}_t) & \underline{g} &= sigm(U_g\tilde{h}_{t-1} + W_g\tilde{x}_t) \\
c_t &= \underline{f} * c_{t-1} + \underline{g} * \underline{i} & h_t &= \underline{o} * tanh(c_t)
\end{aligned}$$

Let $\mathbf{W} = \{W_i, W_f, W_o, W_g\}$ and $\mathbf{U} = \{U_i, U_f, U_o, U_g\}$ be weight matrices of a LSTM. h_t is a column vector that is the output at time t. Similar to the feed-forward neural network case, h_t can be calculated by zeroing out ith column of each $W \in \mathbf{W}$,

if the ith component of $\tilde{h}_t$ is 0. Similarly, when the ith component of $\tilde{x}_t$ is 0, ith column of $U \in \mathbf{U}$ can be zeroed out for computing $\tilde{x}_t$. While evaluating gradients for each $W \in \mathbf{W}$ and each $U \in \mathbf{U}$, the entries that are zeroed out always have gradients of 0. When the dropout probability P is sufficiently large, the resulting gradient vector can be sparse.

6.2 Asyn^{d}-DDPG

Based on DDPG algorithm [50], this section proposes an asynchronous actor-critic method, Asyn^{d}-DDPG, for learning mapless collision avoidance behavior with worker agents. The proposed method enables multiple worker agents asynchronously to optimize a shared policy and q-function. The key to the proposed method is maintaining the sparsity of the gradients applied to the shared policy and q-function. To this end, each agent maintains up-to-date copies of the shared policy and q-function in its local memory and apply independent Dropout regularization to those copies. Figure 6.1 shows the overview of Asyn^{d}-DDPG.

6.2.1 Shared Policy with Dropout Regularization

Considering robotic applications in practice, at each time step, an agent perceives a high-dimensional feature vector from its surrounding environment through an on-board sensor (e.g., a camera or LiDar). The limited sensing capability makes the environment where the agent operates partially observable. To overcome the partial observability, in Asyn^{d}-DDPG, a state of an agent contains a sequence of sensor observations perceived the past l time steps. Also, the state of an agent also contains other information, including locations and velocities. The shared policy and q-function are represented as deep recurrent neural networks with dropout regularization. Their structures are presented in Chapter 4.

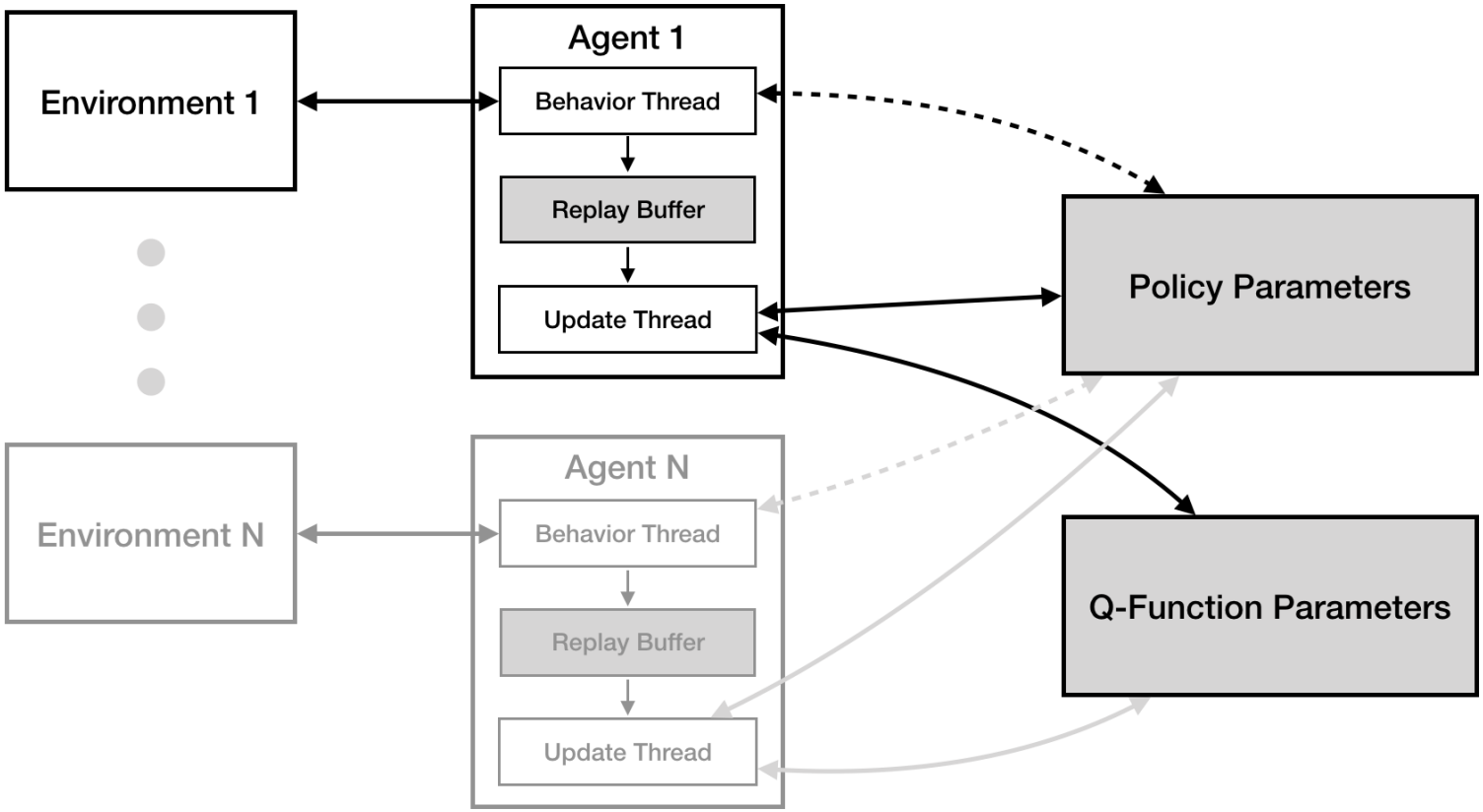


Figure 6.1: The overall workflow of Asyn^{d}-DDPG.

In Asyn^{d}-DDPG, agents independently apply Dropout regularization to the local copies of the shared policy and q-function. This allows agents to evaluate sparse gradients for optimizing both functions during training independently. Note that agents do not apply Dropout to the input of LSTM, since at each optimization step, the states input to the local policy and q-function of an agent must always be the same.

6.2.2 Asynchronous Update

Algorithm 5 summarizes the asynchronous update procedural, followed by all agents. $\mu(\theta_i^t)$ and $Q(w_i^t)$ are the agent i's local copies of the shared policy and q-function. At the time t, the worker agent i evaluates Deterministic Policy Gradients and Temporal Difference Gradients. Since those evaluated gradients are sparse, agents can follow the Hogwild! Strategy [68] to asynchronously apply their gradients to the shared policy and q-function.

Because of the copying mechanism, each worker agent can transfer their knowledge learned in different tasks to the other agents during training. This allows each work agent to improve the shared policy upon the work done by the other agents, which is essential for the agents to synthesize their behavior into a consistent meta policy (i.e., the shared policy). As the gradient evaluation does not depend on task-specific information, the policy jointly learned by all worker agents can be generalized to unfamiliar and potentially more complex tasks.

6.3 Evaluation

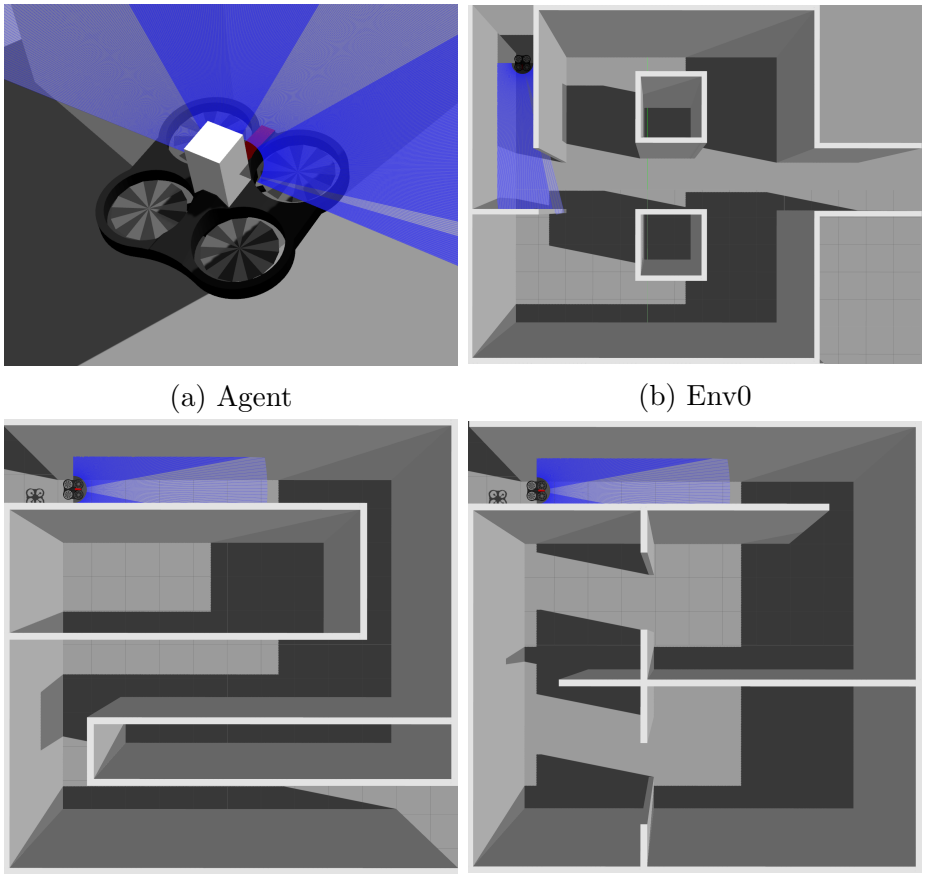
This section demonstrates that Asyn^{d}-DDPG can address the issue of multi-task environments and the issues that have been addressed by RTAC in the previous section. Asyn^{d}-DDPG is the same as the Asynchronous DDPG, except that the

Algorithm 5 The update thread of a worker agent.

```
function AsynUpdate()
    copy w* and θ* to w and θ
    while the shared policy hasn't converge do
        deploy the robot at a fixed initial location
        while an episode is not terminated do
            store experience to replay buffer
            sample a batch of experience
            evaluate policy gradient δ_w
            evaluate temporal-difference gradients δ_θ
            apply δ_w and δ_θ to w* and θ*
            copy w* and θ* to w and θ
```

worker agents utilize Dropout to compute sparse gradients. To demonstrate Dropout's effectiveness, we compare Asyn^{d}-DDPG to the baseline methods, the standard DDPG, and the Asynchronous DDPG, which do not utilize Dropout. In the experiment, the Asyn^{d}-DDPG, the Asynchronous DDPG, and the standard DDPG all follow the RTAC execution architecture. This section shows two parts of the data collected in the experiment. Those two parts of the data show 1) the effect of Dropout on learning the overall multi-task policies, 2) and the effect of Dropout on the learning performance of the worker agent in each training environment. Besides, the evaluation of the multi-task policy learned by Asyn^{d}-DDPG based on the balance of ploy performance on learned tasks and generalizability of the learned policies for unseen environments.

For evaluating Asyn^{d}-DDPG, a complex navigation environment is decomposed into simpler ones. This decomposition is based on the observation that many real-world environments consist of repeating patterns. For example, a building floor can contain doors, hallways, and rooms. This part of the experiments considers that a complex navigation task requires an agent to have three different types of sub-tasks, 1) entering a room, 2) following a hallway, 3) entering a hallway. The Ardrone agent model is used for experimentation. The experiments are based on three environments that represent those sub-tasks.



(a) Agent (b) Env0

(c) Env1 (d) Env2

Figure 6.2: The local patterns used for training. For each local pattern, a circle is a initial location, and a star is a goal.

6.3.1 Simulation Details

Agents: Worker agents are simulated as unmanned aerial vehicles using ROS and Gazebo. A worker agent observes an environment through a LiDar sensor, and it scans the area in its front with 360, evenly spaced lasers at a constant rate. An agent detects a collision if any laser measures a range of less than 0.2m.

Tasks: In experiments, worker agents learn three navigation tasks in maze-like environments, as presented in Figure 6.2. In each environment, a worker agent has a pre-defined initial location and goal. It terminates an episode of navigation if it collides with an obstacle or reaches its goal. The policy jointly learned by all worker agents needs to master navigation tasks in all environments.

6.3.2 Relating Simulation to Theoretical Formulation

The experiments conducted in the chapter are based on the same theoretical formulation as what is in Chapter 5. Unlike the work presented in the previous chapter, we apply the DDPG algorithm under an asynchronous setting. Worker agents use DDPG to compute gradients based on states, actions, and rewards collected from different environments. Specifically, the policy and q-function leaned by worker agents have same neural network architecture as described in Chapter 5, and each worker agent computes its gradients according equation 5.1 and 5.2.

In experiments, we simulate worker agents as aerial vehicles (i.e., Airdrone) navigating in a different maze-like environment. The states and actions and rewards of simulated agents follow the definition given in Chapter 4. The simulation involves a Gazebo simulator, which simulates a robot navigation and training program that executes a reinforcement learning method according to the RTAC 5 architecture. The Gazebo simulator and the training program follow the same procedure described in Chapter 5, to transfer simulation data for reinforcement learning execution. During training, the training program uses a separate thread to compute gradients based on each simulated agent's states, actions, and rewards. Unlike the simulation setting in the previous chapter, each of the threads applies the neural network regularization technique, Dropout[81], to sparse computer gradients for asynchronous stochastic gradient optimization.

6.3.3 Learning Performance

In the experiment, worker agents use the same configuration of dropout rates: 0.4 for LSTM and 0.2 for fully connected layers. For both shared policy and q-function, each fully connected layer has 1024 neurons with ReLu activation functions, and the hidden state of LSTM has the size of 128. Figure 6.2 summarizes the learning performance of Asyn^{d}-DDPG, in comparison with the standard DDPG and the Asynchronous

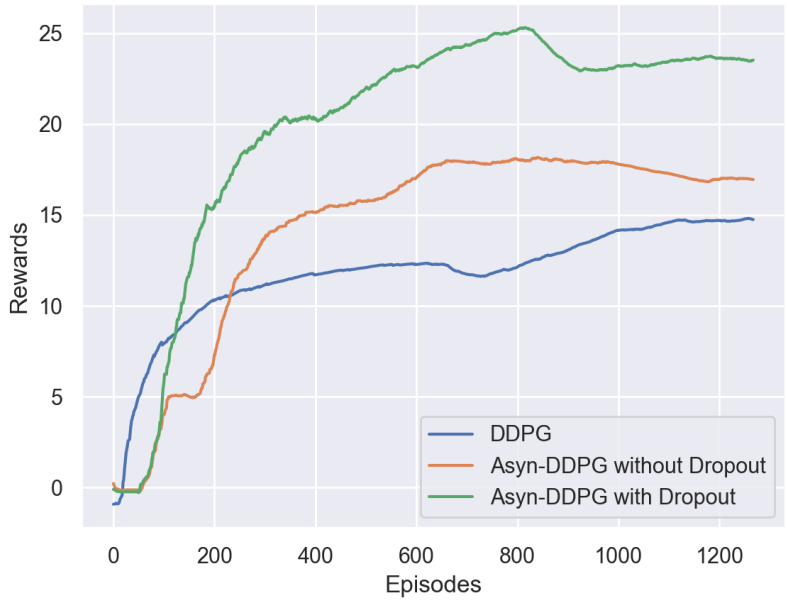


Figure 6.3: Learning performance of DDPG and Asyn^{d}-DDPG in terms of episodic rewards. Both methods are evaluated in Env3.

DDPG (i.e., the Asyn^{d}-DDPG without dropout regularization). The Asyn-DDPG policy with and without dropout regularization is trained in Env0 to Env2. On the other hand, the DDPG policy is trained directly in Env3. For a fair comparison, during training, the separate evaluation agent evaluates the policy jointly learned by the worker agents of Asyn^{d}-DDPG every 5 seconds (in wall-clock time) in Env3. In the Figure, the reward distribution is the moving average of 100 episodic rewards. It shows that the multitask policy learned by Asyn^{d}-DDPG achieves better asymptotic performance compared to the single-task policy learned by DDPG.

To better understand the effectiveness of dropout regularization, the performance of each worker agent of Asyn^{d}-DDPG and Asynchronous DDPG is compared in Env0 to Env2. Figure 6.4 presents the impact of dropout regularization on the Asyn^{d}-DDPG. The results are moving averages of 100 episodic rewards. Based on the results, Dropout can effectively stabilize the performance of Asyn^{d}-DDPG on each

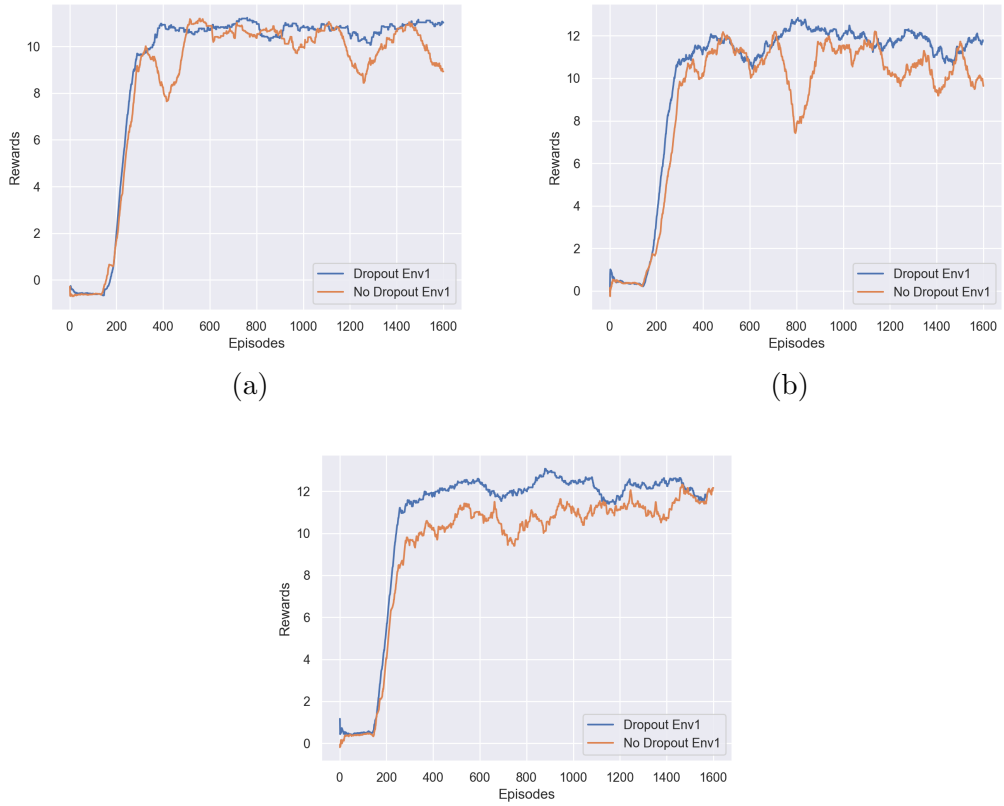


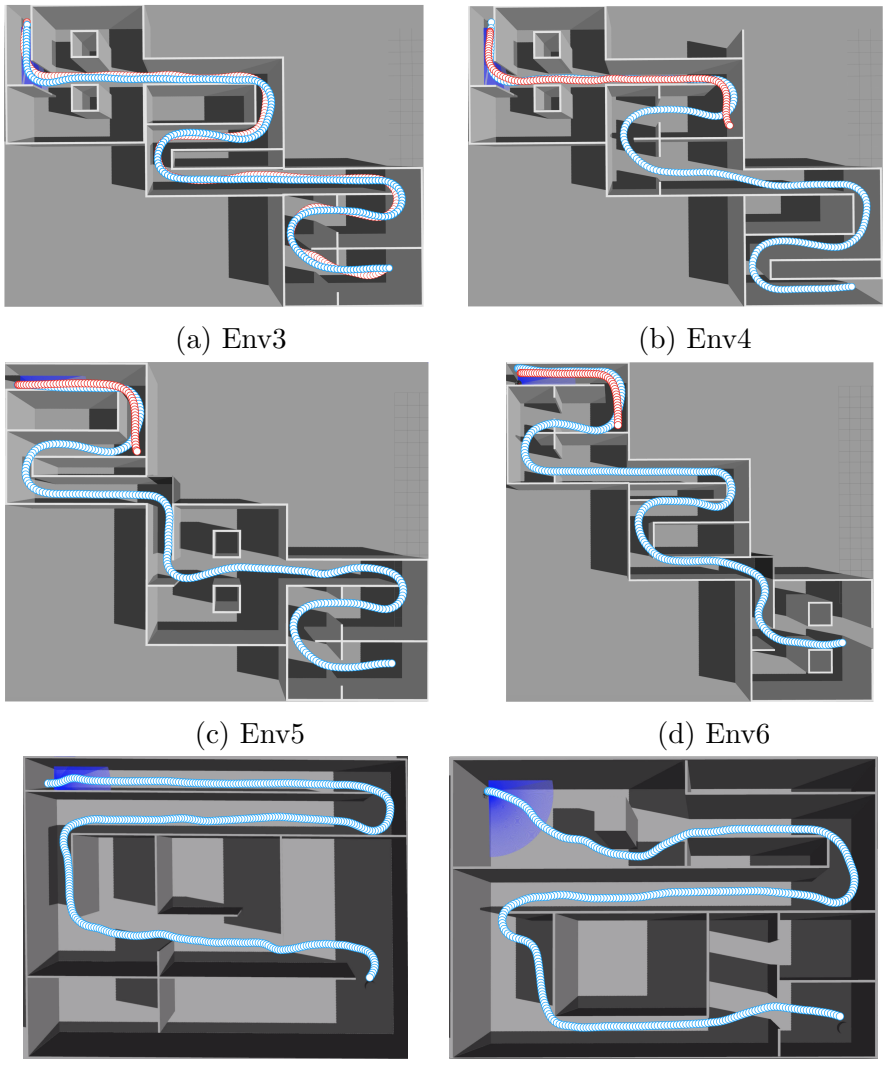
Figure 6.4: Episodic reward distributions based on moving averages for learning navigation tasks in Env0, Env1 and Env2.

task. This supports our claim that ensuring the sparsity of gradients through Dropout regularization can resolve the interference of learning competing tasks.

6.3.4 Policy Performance

This section evaluates the policy learned by Asyn^{d}-DDPG in two aspects: 1) if the learned policy performs equally well in training tasks; 2) if the learned multitask policy has better performance than task-specific policies learned by DDPG.

With balanced performance, the multitask policy jointly learned by all worker agents is expected to complete navigation tasks in all training environments. The first aspect of the evaluation measures the performance of a policy learned by Asyn^{d}-



(a) Env3 (b) Env4

(c) Env5 (d) Env6

(e) Env7 (f) Env8

Figure 6.5: The trajectories planned by an agent using the multitask policy based on Env0, Env1 and Env2. Video demo: `https://youtu.be/4TS5nDlku_g`

DDPG in different concatenations of a training environment. Those environments are presented in Figure 6.5a to 6.5d, together with the trajectories planned by both Asyn^{d}-DDPG and DDPG policies. Those DDPG policies are trained in each of those environments separately, while the multitask the Asyn^{d}-DDPG learns policy in Env0 to Env2. Table 6.1 summarizes the performance of the trained policies during 50 episodes.

	Env3	Env4	Env5	Env6	Env7	Env8
succ_rate	0.88	0.90	0.90	0.90	0.86	0.92
time_step	310.18	310.22	316.71	331.51	415.23	408.31

Table 6.1: The performance of a Asyn^{d}-DDPG policy in all test environments based on 50 episodes. The metrics include the success rates of navigation and the average of the time steps for completing navigation.

According to the results, the multitask policy learned by Asyn^{d}-DDPG can complete the navigation task in all test environments with sufficiently high success rates. The experiment results provide consistent evidence showing that the multitask policy learned by Asyn^{d}-DDPG has balanced performance in all training tasks. Without balanced performance, the learned to multitask policy will result in collisions in one of the training environments during navigation. Besides, the multitask policy always achieves better performance in all environments than the policies learned by DDPG. This supports that the policies learned by Asyn^{d}-DDPG have better asymptotic performance than the single-task policies learned by DDPG.

6.3.5 Policy Generalization

The generality of the policy learned by Asyn^{d}-DDPG is evaluated in two other environments unseen by worker agents during training. Those environments are presented in Figure 6.5e and 6.5f, together with the trajectories planned by the Asyn^{d}-DDPG policy. Table 6.1 shows the success rates of the multitask policy in all test environments (i.e., Env3 to Env8) based on the average performance of 50 episodes. According to the results, the multitask policy learned by Asyn^{d}-DDPG can be well generalized to handle unfamiliar tasks.

6.4 Summary

This chapter presented an asynchronous multitask reinforcement learning method, Asyn^{d}-DDPG, for mapless collision avoidance. By incorporating Dropout regularization, Asyn^{d}-DDPG was able to effectively resolve the interference among worker agents. It enabled the worker agents to asynchronously learn a multitask policy which has balanced performance on learned tasks and good generalizability for unfamiliar tasks. The experiments evaluate Asyn^{d}-DDPG in collision avoidance tasks based on realistically simulated robots and physics-enabled environments. Although the number of tasks used in our experiments is small, we define each task based on a real-world setting. We extensively evaluated the multitask policy learned by Asyn^{d}-DDPG, and our evaluation provided the first step to understand the effects of Dropout regularization on asynchronous multitask reinforcement learning. In future work, we will investigate the impact of Dropout regulation on asynchronous multitask reinforcement learning in a more significant number of procedurally generated task environments. As another direction of future work, we would combine Dropout regularization with strategically adapted learning rates of worker agents, in order to learn a multitask policy with better generalizability.

Chapter 7

Multi-Agent Mapless Collision Avoidance

In practice, there can be multiple robots navigating in the same environment, in order to cooperatively accomplish a task. For example, the multi-robot exploration tasks studied in Chapter 3 require multiple mobile robots to collaboratively build an environment map by constantly navigating in an unknown environment. The reinforcement learning methods presented in the previous chapters focused on mapless collision avoidance for single-robot navigation. This chapter further addresses the issue of enabling mapless collision avoidance for multi-robot navigation through reinforcement learning. In this chapter, we continue referring a robot as an agent, in order to be consistent with existing multi-agent reinforcement learning literature.

The reinforcement learning methods presented in the previous chapters are essentially end-to-end reinforcement learning. Through end-to-end reinforcement learning, a robot maps raw sensor input to control commands using a single neural network. It avoids requiring hand-crafted features for representing robot states and actions. End-to-end reinforcement learning was recently made popular by its successful applications to Atari games [57], Go [78] and continuous control [50]. However, those

remarkable results are all achieved in single-agent domains. Except for [85, 26], there is little work for scaling end-to-end reinforcement learning to multi-agent settings.

In this chapter, we seek to apply end-to-end reinforcement learning to multi-agent mapless collision avoidance. In a robotic application, an agent usually operates in a partially observable environment and perceives high-dimensional sensor data. This poses great challenges to end-to-end multi-agent reinforcement learning. The interaction among agents makes a non-stationary environment [52] and causes multi-agent learning to be unstable. Considering partial observability, multi-agent reinforcement learning needs to map the joint observation space of agents to their joint action space, either through individual policies or a joint policy [93]. When the observation space of each agent is high-dimensional, without effective techniques, the curse of dimensionality can make the learning in a joint observation space intractable.

Addressing the above non-stationary environment issue, Lowe et al. [52] proposed a multi-agent actor-critic method, MADDPG, to learn individual policies for continuous control in a partially observable environment. MADDPG follows the paradigm of centralized training with decentralized execution. During training, each agent optimizes its policy based on joint observation and actions. At execution time, an agent uses its policy to determine actions based on local observation. Although MADDPG can learn optimal policies in the environments where single-agent reinforcement learning is hardly applied, its scalability to high-dimensional observation spaces has not been explored. Extending MADDPG, Rec-MADDPG is proposed, which learns individual end-to-end policies for mapless multi-agent collision avoidance in high-dimensional observation spaces.

Rec-MADDPG categorizes agent observation as interior and exterior observation. Interior observation describes an agent's properties, such as positions and velocities of the agent. On the other hand, exterior observation is the sensor data that the agent perceives from its surrounding environment. Partial observability is related

to the exterior observation of an agent to be a sequence of sensor observations that the agent perceived in the past. To project joint exterior observation of agents into low-dimensional features, we propose two embedding mechanisms, independent and joint embedding, based on recurrent neural networks. In both mechanisms, agents learn their joint exterior observation embedding through end-to-end training. For training efficiency, Rec-MADDPG incorporates parameter sharing and the A3C-based asynchronous framework [55]. This allows Rec-MADDPG to learn a shared policy and q-function with multiple sets of agents grouped in different environments.

We evaluated Rec-MADDPG based on realistically simulated robots in physics-enabled maze-like environments. In experiments, agents need to learn multiple coordinated behaviors for safe navigation in mazes with different layouts. Through experimentation, we demonstrated that Rec-MADDPG could learn optimal end-to-end policies for multi-agent continuous control in the proposed tasks. In contrast, MADDPG was not able to learn effective policies. In extensive experiments, policies learned by Rec-MADDPG with both proposed embedding mechanisms allow agents to navigate all environments safely. Also, compared to joint embedding, independent embedding enabled Rec-MADDPG to learn even better optimal policies.

7.1 Multi-Agent Coordination with Partial Observability

Rec-MADDPG learns coordination among agents that perceive high-dimensional onboard sensor data and operate in continuous action spaces. The multi-agent coordination problem is formulated as a Markov Game defined as

$$(\mathbf{N}, \mathbf{S}, \mathbf{A}, \mathbf{P}, \mathbf{R})$$

$\mathbf{N}$ is a set of agents. $\mathbf{S} = \{S_1, \ldots, S_n\}$ and $\mathbf{A} = \{A_1, \ldots, A_n\}$ is state spaces and action spaces of all agents in $\mathbf{N}$ respectively. Let $\mathscr{S} = S_1 \times \cdots \times S_n$ and $\mathscr{A} = A_1 \times \cdots \times A_n$ be the joint state and action spaces of all agents. S_i and A_i are the state and action spaces of agent i. $\mathbf{P} : \mathscr{S} \times \mathscr{A} \times \mathscr{S} \rightarrow [0, 1]$ is a state transition function. $\mathbf{R} = \{R_1, \ldots, R_n\}$ is a set of reward functions of all agents, and $R_i : \mathscr{S} \times \mathscr{A} \rightarrow \mathbb{R}$ is the reward function of agent i. In a Markov game, an environment is assumed to be Markovian under the joint actions of all agents. Given a joint action, a Markov Game assumes that individual actions are taken at the same time.

In a real-world application of multi-agent coordination, agent i usually has the partial observation of an environment and perceives its state $s_i \in S_i$ through on-board sensors. The agent's state s_i consists of two types of observation, exterior, and interior observation. The exterior observation is about the task environment where the agent situates (e.g., obstacle locations observed by the agent). The interior observation is related to the agent's own properties (e.g., the agent's locations and orientations). Based on the state representation introduced in Chapter 4, in this thesis, an agent i has its interior observation as (p_i^t, g_i) and its exterior observation at time step t as $\bar{o}_i^t = (o_i^{t-l+1}, \ldots, o_i^t)$ o_n^t is the sensor observation perceived by agent i at time step t, and l is the length of the sequence. In this chapter, o_i^t is high-dimensional data that an agent perceived through its on-board sensor.

7.2 Rec-MADDPG

Rec-MADDPG learns individual policies for agents based on the low-dimensional embedding of their joint observation. The proposed method is an extension of MADDPG [52] and follows the framework of centralized learning with decentralized execution.

The key to Rec-MADDPG is representing the exterior observation $\bar{o}_i$ of each agent i into low-dimensional embedding through recurrent neural networks, as (p_i, g_i) is

low-dimensional vector. Let $\mu_i(\theta_i)$ and $Q_i(w_i)$ be the deterministic policy and the q-function of agent i. θ_i and w_i are the parameter vectors of $\mu_i(\theta_i)$ and $Q_i(w_i)$ respectively. Let ϕ be a parameterized non-linear function which projects $\bar{o}_i$ to a low-dimensional feature vector. For each agent i, Rec-MADDPG learns optimal $\mu_i(\theta_i^*)$ using stochastic gradient ascent based on Deterministic Policy Gradients [79]

$$\nabla_{\theta_i} J(\theta_i) = \mathbb{E}[\nabla_{\theta_i} \mu_i(\tilde{s}_i; \theta_i) \nabla_{a_i} Q_i(\tilde{\boldsymbol{s}}, \boldsymbol{a}; w_i) \mid_{a_i = \mu_i(\tilde{s}_i; \theta_i)}] \tag{7.1}$$

$\boldsymbol{a}$ is the joint action of all agents. $\tilde{s}_i = (\phi(\bar{o}_i), p_i, g)$ is the embedded observation of agent i. $\tilde{s}$ is embedded joint observation of all agents. $\tilde{\boldsymbol{s}}$ is the embedded joint observation of all agents. Since the joint exterior observation in $\boldsymbol{s}$ can be extremely high-dimensional, embedding $\boldsymbol{s}$ is the most crucial part of Rec-MADDPG. To this end, two embedding mechanisms are proposed:

$$\tilde{\boldsymbol{s}} = (\phi(\bar{o}_1, \ldots, \bar{o}_n), p_1, \ldots, p_n, g_1, \ldots, g_n) \tag{7.2}$$

or

$$\tilde{\boldsymbol{s}} = (\phi(\bar{o}_1), \ldots, \phi(\bar{o}_n), p_1, \ldots, p_n, g_1 \ldots, g_n) \tag{7.3}$$

The embedding provided by Equation 7.2 and 7.3 are termed as joint embedding and independent embedding. Both types of embedding allow Rec-MADDPG to learn effective policies.

Rec-MADDPG learns the q-function of each agent i using stochastic gradient decent based on Temporal Difference (TD) gradients.

$$\nabla_{w_i} L(w_i) = \mathbb{E}[\nabla_{w_i}(r_i + \gamma Q_i(\tilde{\boldsymbol{s}}', \boldsymbol{a}'; w_i) - Q_i(\tilde{\boldsymbol{s}}, \boldsymbol{a}; w_i))] \tag{7.4}$$

$\tilde{\boldsymbol{s}}'$ is the embedded joint observation which is resulting from agents taking their joint actions. r_i is the reward received by agent i. $\boldsymbol{a}$ and $\boldsymbol{a}'$ are the joint actions taken in

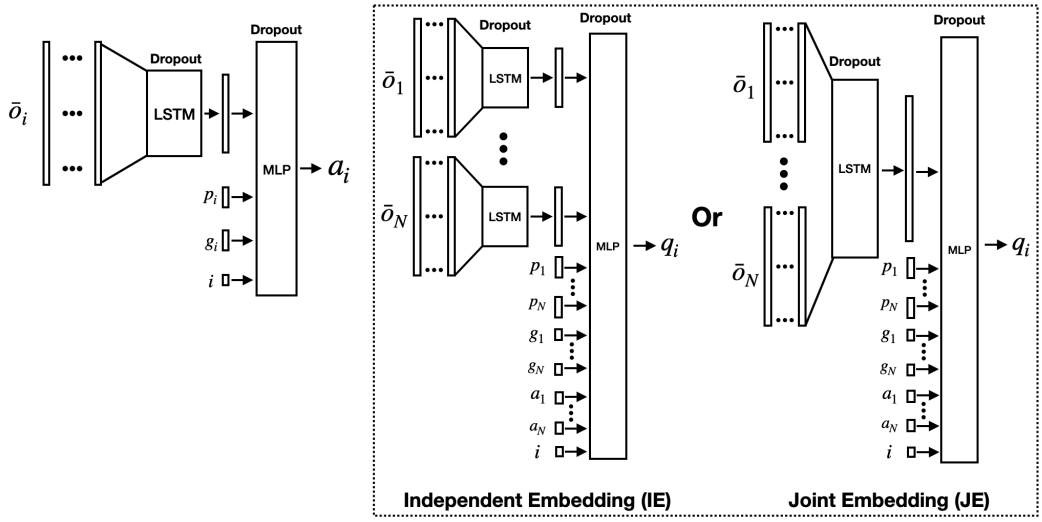


Figure 7.1: left: policy; right: q-function with independent and joint embedding mechanisms.

$\tilde{\boldsymbol{s}}$ and $\tilde{\boldsymbol{s}}'$. In both Equations 7.1 and 7.4, $\mathbb{E}$ is the expectation of the gradients over a batch of joint state-action transactions.

7.2.1 Parameter Sharing & Asynchronous Learning

In Rec-MADDPG, the policy and the q-function of each agent are represented as deep neural networks which embed exterior observation of the agent using LSTM [31] components. For training efficiency, Rec-MADDPG incorporates parameter sharing. In this case, all agents share a single policy and a single q-function.Figure 7.1 shows the neural network representation of the shared policy and q-function.

Rec-MADDPG learns the joint and independent embedding through a single LSTM component. In terms of the joint embedding, the exterior observation of all agents is concatenated before being embedded by LSTM. As for the independent embedding, all agents' exterior observation is input to the same LSTM in a sequence and then concatenated. Both types of embedding are input to the shares q-function to determine the q-values of an agent in the context of the other agents. Rec-MADDPG learns both types of embedding through end-to-end training of the shared q-function.

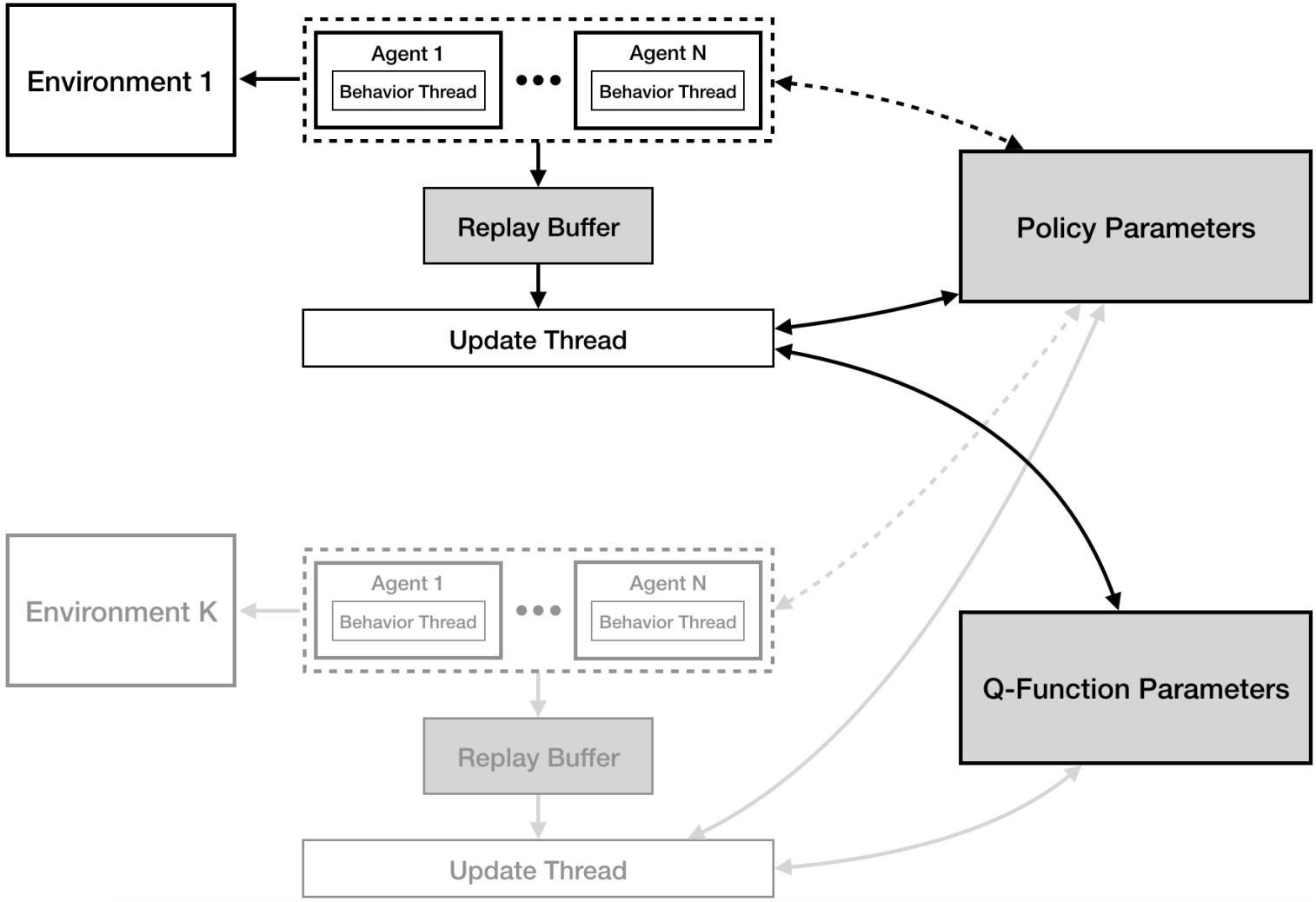


Figure 7.2: The parallelized training paradigm of Rec-MADDPG

In a parameter-sharing setting, an agent i determines its actions and q-values by augmenting its input to the shared policy and q-function with its own ID i. Rec-MADDPG utilizes the asynchronous framework proposed in single-agent reinforcement learning [55] as an asynchronous training paradigm.

Figure 7.2 illustrates the proposed paradigm, where multiple sets of agents are grouped in different environments and asynchronously optimize the globally shared policy and q-function. In the proposed paradigm, each environment contains the same number of agents indexed by the same set of IDs. After each time step, a centralized Manager thread stores their joint state-action transactions into a separate replay buffer and optimize the globally share policy and q-function according to equation 7.1 and 7.4. Algorithm 6 summarizes the learning procedure executed by a Manager thread in a particular environment.

It is important to note that the asynchronous framework proposed in [55] does not rely on replay buffers for stable learning. Instead, it allows agents to accumulate their gradients before applying them to the shared policy or q-function. Different from [55], the proposed paradigm relies on replay buffers for asynchronous learning. The reason is that, in the proposed paradigm, accumulated gradients will most likely explore during training. Since agents in the same environment must take their actions simultaneously, the agents have to apply their accumulated gradients to their shared policy and q-function all together at each optimization step. That often leads to a large destructive policy/q-function update. Under the proposed paradigm, empirically replay buffers can well stabilize the asynchronous learning process and also allow Rec-MADDPG to update the shared policy and q-function with more significant learning rates.

Algorithm 6 The update thread of a group of agents.

```
1: function EnvLearn(μ(θ), Q(w))
2:     initialized s_i = (ō_i, p_i, g_i) for each agent i
3:     for time step t = 1 to T do
4:         for each agent i do
5:             take action a_i = μ(s_i, i; θ)
6:             observe s'_i = (ō'_i, p'_i, g) and receive reward r^i
7:             s_i ← s'_i
8:         store (s, r^e_0, ..., r^e_n, a, s') to D^e
9:         for each agent i do
10:            sample a batch from D^e for gradient evaluation
11:            optimize μ(θ) and Q(w) using ADAM
```

7.3 Evaluation

Rec-MADDPG extends the Asyn^{d}-DDPG algorithm and conducts asynchronous learning based on agents' joint states and actions. Based on the experiment results of RTAC and Asyn^{d}-DDPG, this section demonstrates that Rec-MADDPG can further address the issue of multi-agent interaction and learn individual policies for interacting agents in a cooperative environment. This successfully demonstrates that Rec-MADDPG can address all the five issues that may arise in real-world reinforcement learning applications, including partial observability, high-dimensional sensor data, real-time operation, multi-task environments, and multi-agent interaction. In addition, the multi-agent policy learned by Rec-MADDPG is evaluated in terms of its effectiveness on learned tasks and generalizability for unseen environments.

In the experiment, the Rec-MADDPG follows the same execution architecture as what the Asyn^{d}-DDPG follows (i.e., RTAC), and it also utilizes Dropout to compute sparse policy gradients for asynchronous learning. Different from Asyn^{d}-DDPG, Rec-MADDPG uses the MADDPG algorithm [52] to conduct the asynchronous learning based on joint states and actions of agents. Extending the standard MADDPG, Rec-MADDPG projects the joint states of agents into a low-dimensional embedding through recurrent neural networks. To this end, two embedding mechanisms, the joint

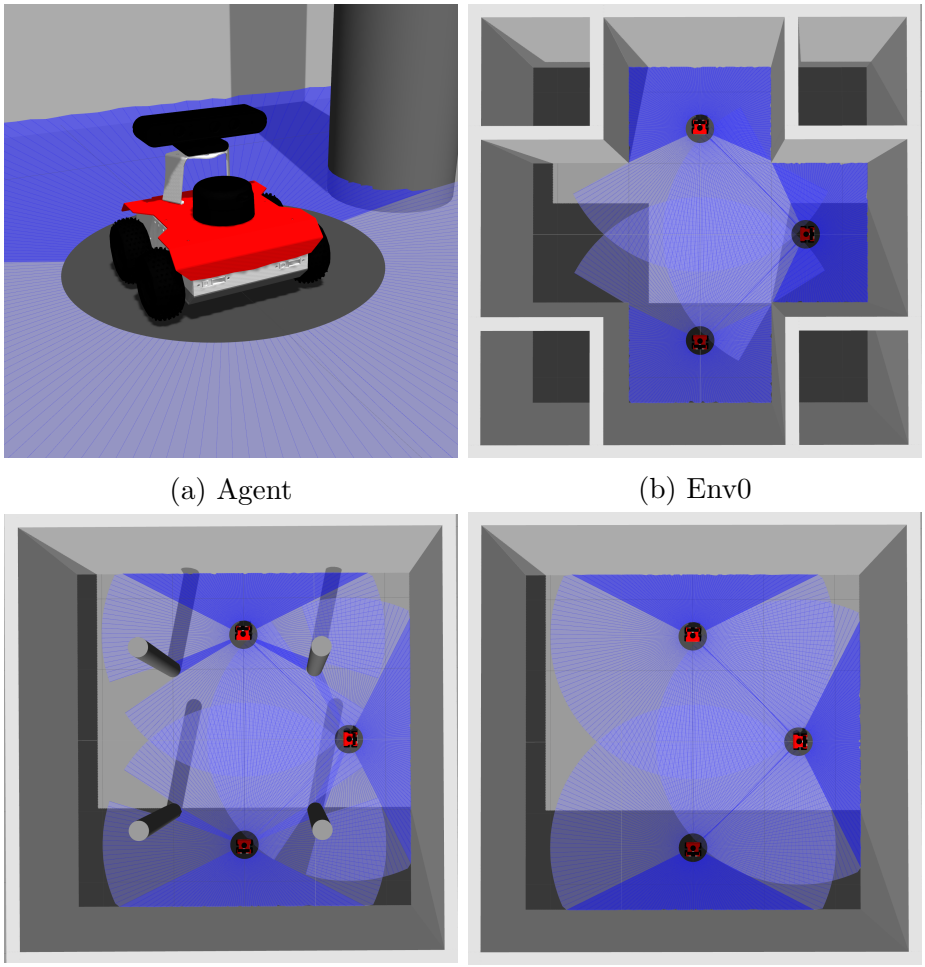
embedding, and the independent embedding are proposed. To show that those two embedding mechanisms can enable Rec-MADDPG to learn better optimal policies for multi-agent navigation, Rec-MADDPG is compared with the standard MADDPG. In the experiment, MADDPG follows the same asynchronous training mechanism adopted by the Rec-MADDPG and also utilizes Dropout to compute sparse gradients for asynchronous learning. Such an asynchronous reinforcement learning setup is also essentially the same as the one used in the Asyn^{d}-DDPG experiments so that the experiments in this section can be consistent with the experiments in the previous sections.

7.3.1 Simulation Details

In experiments, agents with continuous actions must navigate to their pre-defined goals based on their local sensor observation without collisions. To address the issues that may arise in real-world robotic applications, the implementation of experiments is based on ROS and Gazebo, which provide realistic robot simulation and physics-enabled environments.

Agents. Agents are simulated based on physical robots, i.e. ROSbot 2.0. Figure 7.3a presents an simulated agent in our experiments. Each agent observes its surrounding environment using a range sensor that measures the distance between the agent and its nearby objects using evenly spaced outgoing rays. Each time step, each agent uses its range sensor to observe a vector of 180 ranges, each of which determines the agent's distance to an obstacle detected by a ray.

Environments. For successful completion of the proposed task, agents must learn their behavior in the presence of obstacles with various distributions. Rec-MADDPG learns a shared policy for agents to master the proposed navigation task in all sub-environments presented in Figure 7.3b to 7.3d.

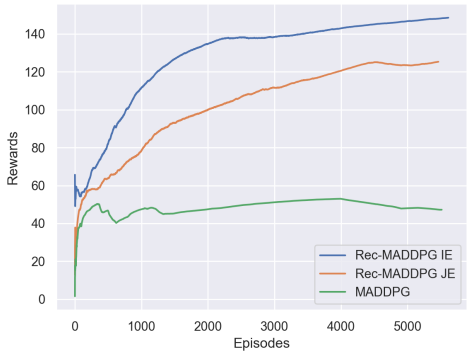


(a) Agent (b) Env0

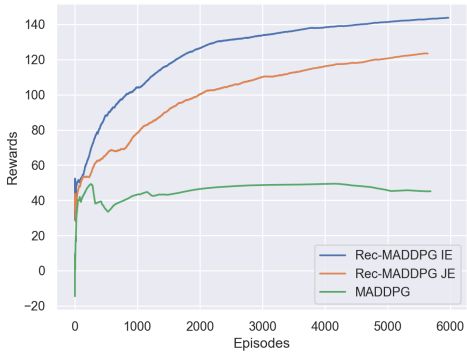
(c) Env1 (d) Env2

Figure 7.3: The agent and environments used in experiments.

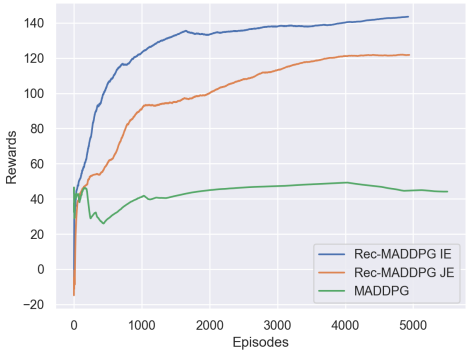
In each environment, each agent is assigned a pre-defined goal where it needs to navigate. An agent reaching its goal, if its distance to its goal is lower than 0.5 (i.e., the diameter of a simulated robot). During navigation, each agent must avoid collisions with both obstacles and the other agents. If an agent reaches its goal early, it must learn to hover its goal while waiting for the other agents to reach theirs. For each environment, agents complete their tasks successfully if they all reach their goals without collisions.



(a) Rewards in Env0



(b) Rewards in Env1



(c) Rewards in Env2

Figure 7.4: Episodic reward distributions during training. IE and JE stand for the Rec-MADDPG with independent and joint embedding.

7.3.2 Relate Simulation to Theoretical Formulation

This chapter focuses on learning policies for multi-agent collision avoidance. To support multi-agent reinforcement learning, we extends the simulation setting presented in Chapter 5 and allows multiple agents to navigate in the same environment. In this chapter, we simulate an agent as a land-based vehicle (i.e., ROSbot). The states, actions and rewards of an agent has the definition given in Chapter 4.

The experiments in this chapter is based on a multi-agent reinforcement learning method, MADDPG [52]. This method allows an agent to alternate between optimizing its own policy and q-function through stochastic gradient optimization. Specifically, the agent computes the gradients for optimizing its own policy and q-function

according to equations 7.5 and 7.6.

$$.\nabla_\theta J(\theta) = \mathbb{E}\{\nabla_\theta \log \mu^i_\theta(s^i_t) \nabla_{a^i_t} Q(\boldsymbol{s}_t, \boldsymbol{a}_t) \mid_{a^i_t = \mu^i_\theta(s^i_t)}\} \tag{7.5}$$

$$\nabla_w L(w) = \mathbb{E}[\nabla_{w^i}(r^i_t + \gamma Q^i(\boldsymbol{s}_{t+1}, \boldsymbol{a}_{t+1}; w) - Q^i(\boldsymbol{s}_t, \boldsymbol{a}_t; w))] \tag{7.6}$$

In the above equations, $\boldsymbol{s}_t$ and $\boldsymbol{a}_t$ are the joint states of all agents in the same environment at time t. s^i_t and a^i_t are the state and action of agent i at time t. During training, an agent must have access to the joint state and actions of all agents in the same environment, in order to evaluate its own gradients. To support this in our simulation, we let the Gazebo simulator to transfer the data perceived by all the agents in the same environment to the training program, which will subsequently structure those data as joint states, actions and rewards of those agents and store them in a separate buffer as different transactions, $(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{r}_t)$. $\boldsymbol{r}_t$ is the rewards received by each of the agents in the same environment at time t.

The experiments conducted in this chapter utilizes weigh-sharing technique which allows agents in the same environment to use the same neural networks as their policies and q-functions. Figure 7.1 presents the neural network structure implemented in our experiments. During training, a separate thread samples multiple transactions, $(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{r}_t, \boldsymbol{s}_{t+1}, \boldsymbol{a}_{t+1})$, from the buffer which stores the joint state, action and rewards of agents. For each sampled transaction, the thread will compute the gradients for each agent i according to equation 7.5 and 5.2 without the expected value notation $\mathbb{E}$. For each agent i, the thread computes the expected values of the agent's gradients by averaging those computed gradient values with respect to each of the parameters of the shared policy and q-function. Then, the thread will apply the expected gradients of each agent to the parameters of the shared policy and q-function.

The proposed methods will requires asynchronous reinforcement learning with multiple groups of agent in different environments 7.2.1. In this case, the Gazebo

simulator will transfer the data perceived by different groups of agents to the training program. Then, the training program will structure the data of each group of agents as separate transactions of joint state, action and rewards. Each transaction is stored into separate buffers. For each buffer, a separate thread will compute expected gradient values according to the above procedure. The thread will then asynchronously applies those computed gradients to the shared policy and q-function.

7.3.3 Learning Performance

With both independent and joint embedding, the performance of Rec-MADDPG is compared with the standard MADDPG [52]. The evaluation of Rec-MADDPG and MADDPG uses asynchronous RTAC as execution architecture. In experiments, both methods need to learn a shared policy that can coordinate agents in all three environments presented in Figure 7.3b to 7.3d.

In terms of the shared q-function, the embedding of the joint exterior observation of all agents, together with other state features, are input to a 3-layer perceptron, where each layer has 2048 neurons. As for the shared policy, the exterior observation of an agent is also embedded by a LSTM component and then input to a 3-layer perceptron, where each layer has 1024 neurons. ReLu is used as an activation function for both q-function and policy. For comparison, LSTM is eliminated from the shared policy and q-function in MADDPG. Instead, MADDPG flattens the exterior observation of agents into a one-dimensional vector.

Figure 7.4 presents the experiment results. In all three environments, Rec-MADDPG significantly outperforms the MADDPG. Based on the results, both independent and joint embedding can enable Rec-MADDPG to learn policies that can effectively coordinate agents in all three environments. Compared to joint embedding, independent embedding enables Rec-MADDPG to achieve considerably

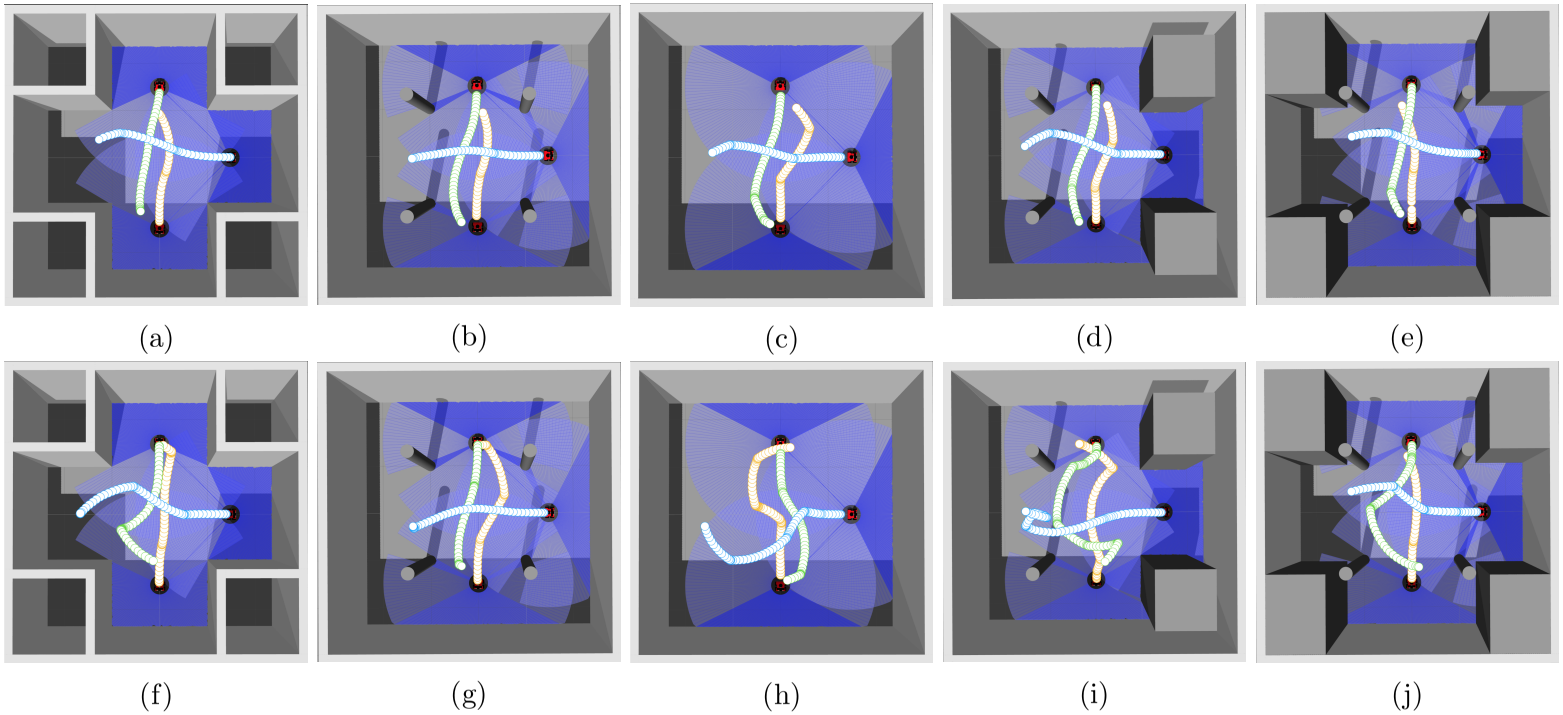


Figure 7.5: Trajectories planned by the policies learned through Rec-MADDPG. (a)-(c): Rec-MADDPG with independent embedding; (d)-(f): Rec-MADDPG with joint embedding. Video demo: `https://youtu.be/UqMvFdcCCG4`

higher rewards. The reason is that independent embedding can reduce the joint observation space of agents.

With joint embedding, Rec-MADDPG learns the embedding in the joint exterior observation space with the sized of $|\bar{o}|^n$. $|\bar{o}|$ is the dimensionality of an agent's exterior observation, and n is the number of agents. On the other hand, independent embedding enables Rec-MADDPG to learn exterior observation embedding for each agent independently through a shared LSTM. That reduces the size of agents' joint exterior observation space from $|\bar{o}|^n$ to $n|\bar{o}|$. With smaller joint exterior observation space, it's easier for Rec-MADDPG to learn effective embedding during training.

7.3.4 Policy Performance

In this part of experiments, the performance of the policies learned by Rec-MADDPG and MADDPG are compared in each of the three environments presented in figure 7.3b to 7.3d. To show the detailed performance of policies learned by Rec-MADDPG, a team of agents use the learned policies to navigate in each of the three environments for 120 episodes. Figure 7.5 presents example trajectories planned by the policies learned through Rec-MADDPG and MADDPG.

	Rec-MADDPG IE			Rec-MADDPG JE		
	time_step	succ_rate	min_dist	time_step	succ_rate	min_dist
Env0	48.611	0.824	0.532	63.032	**0.917**	0.716
Env1	45.883	0.642	0.591	57.576	**0.972**	0.753
Env2	53.472	**0.982**	0.653	68.121	0.509	0.634
Env3	52.770	**0.951**	0.580	133.08	0.174	0.881
Env4	46.090	**0.803**	0.499	74.218	0.653	0.709

Table 7.1: Summarized metrics of navigation trajectories planned by agents using learned policies in all environments. The minimum distance between robots (i.e., min_dist) is measured in meters.

Table 7.1 summarizes metrics of the trajectories planned agents in all three environments, including the success rates and the time steps, and minimum distance

between robots. Based on the results, Rec-MADDPG, with both independent and joint embedding, can learn policies that enable agents to complete their navigation in all three environments. On the other hand, MADDPG can not learn effective policies at all. However, for each embedding mechanism, the learned policies have considerably low performance in one of the three environments, while having a strong performance in the other two. This result suggests that both embedding mechanisms were able to compliment each other in certain scenarios.

7.3.5 Policy Generalization

The generalizability of the learned policies is evaluated in unfamiliar environments. Those trajectories suggest that independent embedding enabled Rec-MADDPG to learn policies with significantly better performance in unfamiliar environments.

7.3.6 Learning Policies with Improved Reward Function

In the chapter, Rec-MADDPG learns policies for multi-robot collision avoidance based the reward function below:

$$R = \begin{cases} -1 & \text{collided,} \\ 1 & \text{reached the final goal,} \\ max(0,\ d' - d) & \text{otherwise.} \end{cases} \tag{7.7}$$

The d and d' are the euclidean distance from an agent's previous and current locations to its goal. Indeed, the above function was also the reward function used by the reinforcement learning methods presented in Chapter 5 and 6.

In general, this reward function allowed agents to learn effective policies for collision avoidance. However, it does not capture all navigation performance metrics, as presented in Chapter 4. This makes us hypothesize that the quality of learned

policies can be improved if we incorporate more relevant metrics into the rewards. This section re-trained the policies with an independent embedding mechanism based on the following reward function:

$$R = \begin{cases} -1 & \text{collided,} \\ 1 & \text{reached the final goal,} \\ (d' - d) - \alpha\frac{1}{c} - \beta\varphi & \text{otherwise.} \end{cases} \tag{7.8}$$

In the above equation, we incorporate the additional metrics, c, and φ. c denotes the minimum distance from an agent's current location to obstacles, and φ is the agent's travel distance within a one-time step. α and β are scalar values serving as coefficients. After incorporating those addition metrics, the reward function would favor shorter trajectories that allow an agent to stay further away from obstacles. We compare the performance of the learned policy within Table 7.2. Those results are summarized based on 120 episodes planned by agents using their learned policies.

	original reward function			improved reward function		
	time_step	succ_rate	min_dist	time_step	succ_rate	min_dist
Env0	48.611	0.824	0.532	61.491	0.852	0.781
Env1	45.883	0.642	0.591	59.862	0.557	0.755
Env2	53.472	0.982	0.653	75.428	0.983	0.956

Table 7.2: The performance of the policies learned based on the original reward function and the improved reward function. The original and improved reward functions are given by equation 7.7 and 7.8 respectively.

According to the results, the improved reward function allows agents to distance each other during navigation significantly. Compared to the original reward function, the improved reward function does not significantly change agent navigation's success rate. In Env1 (as shown in Figure 7.3c), both reward function give low success rate. According to Table 7.1, the joint embedding mechanism achieved the success rate of 0.972 in the Env1. This suggests that the Independent Embedding mechanism is not

well suitable for Env1. The reason can be that the Independent Embedding is not as good as the Joint Embedding mechanism in capturing the relationships among the observation of different robots. As we can observe in the same table, the improved reward caused the robots to take more time to complete their navigation. Because the improved reward function encourages the robots to keep larger distances to obstacles, they would have to reroute their path during their navigation. This caused the robots to plan a longer path to their goals eventually. However, it is important to have the travel distance component in the reward function. With the component, the robots can learn to minimize the lengths of their rerouted paths instead of learning to plan an arbitrary long path to their goals.

7.4 Summary

This chapter presented a multi-agent reinforcement learning method, Rec-MADDPG, which enabled mapless multi-agent collision avoidance for multi-robot navigation. We proposed independent and joint embedding mechanisms based on recurrent neural networks in order to project joint exterior observation of agents to low-dimensional features. By incorporating parameter sharing and an A3C-based asynchronous framework, Rec-MADDPG learned a shared policy and q-function with multiple sets of agents grouped in different environments. We evaluate Rec-MADDPG in robotic navigation tasks based on realistically simulated robots and physics-enabled maze-like environments. In extensive experiments, we demonstrated that Rec-MADDPG could significantly outperform MADDPG. With both independent and joint embedding, Rec-MADDPG learned effective policies for agents to master those navigation tasks. Also, those learned policies were general enough to handle environments that are unseen by agents during training. Compared to joint embedding, independent

embedding could enable Rec-MADDPG to learn even better optimal policies, which allowed agents to plan shorter and smoother trajectories to their goals.

Chapter 8

Conclusions

8.1 Summary

Collision avoidance is a difficult problem, and researchers have been actively studying it under different robot configurations and environment dynamics. In this thesis, we first studied map-based collision avoidance in navigation tasks involving multiple robots. We proposed two multi-robot systems that allowed robots to build environment maps for navigation under centralized and distributed settings. In this study, robots following different pre-planned paths may encounter each other during navigation and must avoid collisions in real-time based on their local sensor observation. This was because the environment map built by the robots was not entirely accurate, and it did not incorporate the locations of those moving robots. Therefore, it is essential for a robot to utilize mapless collision avoidance methods for safe navigation.

Motivated by the study of map-based collision avoidance, the rest of the thesis focused on mapless collision avoidance. For collision avoidance in practice, robots operate in real-time and must meet the real-time constraint. However, the conventional robotic methods for collision avoidance require a robot to infer optimal actions within a predefined time limit at each time step. Such online action inferring can be com-

putationally expensive, particularly when a robot has limited computing resources. This thesis presented a reinforcement learning approach to learn a policy that allows a robot to deterministically map local sensor observation to optimal actions for mapless collision avoidance. By directly mapping sensor observation to actions through the learned policy, a robot can avoid expensive online action inferring and minimize the time delay before acting. After formulating the mapless collision avoidance as a reinforcement learning problem, we proposed a real-time architecture, RTAC, to have robots learn policies through model-free off-policy reinforcement learning under real-time constraints. Based on RTAC, we proposed Asyn^{d}-DDPG for learning multi-task policies for mapless collision avoidance. By extending Asyn^{d}-DDPG, we proposed Rec-MADDPG to incorporate multi-agent interaction into reinforcement learning and learn policies for mapless collision avoidance of multiple robots.

In extensive experiments, this thesis demonstrated the great potential of deep reinforcement learning to solve mobile robots' real-world collision avoidance. We evaluated the proposed work in simulation, which allowed simulated robots to operate under real-world settings. Through experimentation, we showed that RTAC could effectively apply reinforcement learning to real-time control, fundamental for robot navigation in the physical world. Specifically, RTAC effectively reduced the control delay variance and enabled robots to operate in real-time to learn optimal policies. It can also be scaled to support asynchronous reinforcement learning and allow the proposed reinforcement learning methods, Asyn^{d}-DDPG and Rec-MADDPG, to learn optimal policies under real-time constraint. Asyn^{d}-DDPG and Rec-MADDPG addressed multi-task and multi-agent control, respectively, which are important issues for collision avoidance in the physical world. The physical world is an environment with high-level uncertainties, and it can pose unforeseen scenarios where robots need to avoid collisions. In experiments, those proposed methods could learn policies that

have good generalizability and allow a single robot or multiple robots to conduct safe navigation in different scenarios, including those unseen during training.

8.2 Accomplishment

This thesis focused on a reinforcement learning approach for mapless collision avoidance under real-world settings. As a long-standing problem, mapless collision avoidance has been actively studied under robotics research. Conventionally, mapless collision avoidance is formulated as a real-time optimization problem, which requires a robot to search for an optimal its action space (or configuration space) within a pre-defined deadline. With limited onboard computing resources, such online action inferring can be computationally expensive and may fail to meet the deadline. In the thesis, we mitigate the issue of online action inferring by learning deterministic policies through reinforcement learning. With a learned policy, an agent can deterministically select an action based on its sensor observation at each time step, which avoids searching an action space.

In general, applying reinforcement learning to robotics task requires learning policy in a simulated environment and deploying the learned policy in the second phase. In this thesis, we study those issues in a simulated environment, which serves the pre-requisite step of deploying reinforcement learning for mapless collision avoidance in the physical world. Applying reinforcement learning to mapless collision avoidance must deal with various issues in the physical world. In this thesis, we primarily addressed the following issues in the context of reinforcement learning, including partial observability, high-dimensional sensor data, real-time control delay, multi-task environments, and multi-agent interaction.

Historically, reinforcement learning was able to solve small scale problems. The above issues posed a challenging environment for reinforcement learning, and address-

ing all of them at once can be difficult. Instead, we studied different reinforcement learning methods to address the above issues progressively. In the end, our multi-agent reinforcement learning methods, Rec-MADDPG, was able to deal with all those issues. The Rec-MADDPG learns individual policies for each of the robots navigating in the same environment. Compared to the solutions where robots adopt the same controller for collision avoidance, Rec-MADDPG better learns customized policies for collision avoidance under challenging scenarios. However, learning individual policies poses exponentially increasing policy search space, making Rec-MADDPG only applicable to a small number of robots. We found that the convergence time of Rec-MADDPG increases as we increase the number of robots through experiments. However, there is no evidence showing that Rec-MADDPG would not converge if we can give it enough training time.

The Rec-MADDPG is based on the other two reinforcement learning methods we developed in the thesis: the RTAC architecture and Async^{d}-DDPG. Robots navigating in the physical world operates in real-time. To support real-time operation during reinforcement learning, we proposed the RTAC architecture. Through empirical experiments and theoretical analysis, we show that it is crucial to support real-time operations to ensure the Markov property. For learning efficiency, we also designed RTAC to support asynchronous reinforcement learning. Robot navigating in the physical world needs to avoid collisions under different scenarios. To this end, we developed Async^{d}-DDPG as a way to synthesis collision avoidance behavior under different scenarios into a consistent policy through asynchronous reinforcement learning. Both RTAC and Async^{d}-DDPG provides insights about the performance of deep reinforcement learning robot navigation in the physical world. They are the fundamental building blocks for Rec-MADDPG to learn customized collision avoidance policies under real-world settings.

8.3 Discussion

The reinforcement learning approach studied in the thesis focused on collision avoidance of robots with low traveling speed. In our experiments, a robot is subject to a maximum speed. Through reinforcement learning, the robot needs to determine its desired speed at any time steps. The proposed reinforcement learning methods treat the maximum traveling speed as an intrinsic property of a robot. In this case, the policies learned by those methods are for the robots with redefined maximum traveling speed. In experiments, we empirically set such maximum speed to be 1m/s while considering the constraints of robot motion (i.e., kinematic constraints) and navigation environments' characteristics. When multiple robots are involved in an experiment, the robots have the same maximum traveling speed (i.e., 1m/s).

Navigation is ubiquitous among the tasks involving mobile robots. In practice, a robot can travel in speed that is much higher than 1m/s. A robot's travel speed is usually subject to navigation task requirements and obstacle distributions of an environment. For example, in a cluttered indoor environment, a robot would have to move at low speed to avoid collisions, as a robot has to change its heading directions constantly. On the other hand, a robot can also navigate the open outdoor environment, such as traveling on a highway. In this case, the control task would be significantly different from the ones studied in the thesis. The robot's navigation performance shall be measured with additional metrics, such as stabilizes and response time. Addressing the issue of high traveling speed requires an extension of thesis work, e.g., incorporating additional reward function components, computing more effective policy gradients, and designing a more appropriate reinforcement learning mechanism. The work studied in the thesis contributes to learning robust policies for safe navigation in real-world environments.

This thesis progressively studied reinforcement learning methods for enabling multiple robots to avoid collisions in various navigation scenarios under real-world set-

tings. Eventually, we achieve this objective through the proposed multi-agent reinforcement learning method, Rec-MADDPG. This method follows the centralized-learning-and-distributed-execution paradigm. Essentially, it learns separate policies for individual robots based on their joint state and actions. As the joint state and action spaces grow exponentially as the number of robots increases, the Rec-MADDPG is only suitable for learning policies for a small number of robots. In the thesis work, Rec-MADDPG was able to learn policies for three robots navigating in three different scenarios. In our experiments, it took approximately one day to learn the policies for those robots. As we increase the number of robots to five, the Rec-MADDPG seems not going to converge on optimal policies for a single scenario within a reasonable amount of time (i.e., ≥ 2 days). This suggests that learning collision avoidance policies for many robots require a significant extension of Rec-MADDPG.

When multiple robots are navigating in the same environment, it is a rare case when all the robots need to avoid collisions with each other. It is more often for a subset of the robots to encounter each other during their navigation. In this case, it is necessary to have policies for a small number of robots to avoid collisions. To this end, the proposed Rec-MADDPG demonstrates the feasibility of applying reinforcement learning to learn policies for collision avoidance under various obstacle distributions. In extensive experiments, we show that recurrent neural networks can effectively handle the temporal sequence of high-dimensional sensor data and can be utilized to approximate the optimal q-function for high-dimensional joint state space. The Rec-MADDPG provides the opportunities to empirically understand deep reinforcement learning behavior in both high-dimensional joint state space and continuous action space.

8.4 Future Work

Reinforcement learning is generally categorized as model-free and model-based methods. This thesis focused on model-free reinforcement learning for mapless collision avoidance. On the other end of the spectrum, model-based reinforcement learning has also been actively studied in robotics research. Although model-free methods can theoretically achieve the same optimal performance without estimating the model of an environment, model-based methods can achieve better sample efficiency by collecting more informative experience based on an environment model. In future work, we would apply model-based reinforcement learning to mapless collision avoidance. Unlike a model-free method that purely relies on state values, a model-based method allows a robot to predict the outcome of an action based on the estimated model of an environment and select actions based on more complete criteria. Therefore, a model-based method has an opportunity to learn predictable and interpretable behavior for collision avoidance.

Applying reinforcement learning to robotic tasks typically involves learning policy in simulation and transfer the learned policy to a robot operating in the physical world in the second phase. This thesis focused on learning policies under real-world settings in simulation, an important step for a practical application of reinforcement learning. It primarily addressed the following five issues that can arise in real-world reinforcement learning applications: partial observability, high-dimensional sensor data, real-time control delay, multi-task environment, and multi-agent interaction. For future work, there can be other issues related to reinforcement learning in the physical world. It will be worthwhile to incorporate the uncertainties caused by robot localization into reinforcement learning to learn more robust policies for collision avoidance. Besides, a robot can extract semantic information from sensor data (e.g., detecting pedestrians and recognizing traffic signs) and possibly incorporate the information

into its environment model, so that the robot can make more intelligent decisions for collision avoidance.

In some navigation tasks, there can be many robots that need to move through narrow spaces. This can be typical scenarios where robots form congestion, and such scenarios were also demonstrated in the motivating work presented in Chapter 3. Reinforcement learning can be a candidate solution for robots to resolve congestion and ensure safe navigation in narrow spaces. However, the proposed multi-agent reinforcement learning method, Rec-MADDPG methods, is only suitable for learning policies for a small number of robots. For future work, we would extend the Rec-MADDPG to enable collision avoidance among a large number of robots. As demonstrated in an extensive experiment, the proposed independent and Joint embedding mechanism has its advantages on collision avoidance under different scenarios. Based on those two embedding methods, we can design a mechanism that can both well embed observation of each robot and capture the readership among observation of different robots. In terms of learning policies for many robots, learning customized policies for each robot would pose an extremely large policy search space and make learning intractable. To reduce the policy search space, we would effectively fuse the experience collected by different robots and learn a single policy shared by all the robots for collision avoidance. However, the time for learning such a policy can increase drastically, as we have more agents navigate in the same environment. It is important to increase the sample efficiency of the reinforcement learning method. As suggested in recent literature [83, 16, 101], we can let agents use a pre-programmed motion controller to collect more useful experiences. The agents can use those experiences to initialize reinforcement learning. With the pre-programmed controller's guide, agents can discover better optimal policies for collision avoidance.

Navigation tasks can require robots to travel at various speeds. The reinforcement learning approach studied in the thesis is only suitable for robots traveling at

low speed. As autonomous driving and delivery robots become increasingly popular, it is important to enabling collision avoidance of robots traveling at high speed in future work. To support navigation tasks requiring high-speed traveling, we would incorporate the different simulation setup, such as [76, 18], for reinforcement learning. In the thesis, we assumed that all robots have the same maximum travel speed. Considering generalizability, we propose a reinforcement learning method for robots with different maximum traveling speeds. To this end, a learned policy must be subject to the physical constraints of different robots and determine the desired velocities under various maximum speed. To this end, we need to study alternative training paradigms and algorithms for learning policies that can not only work in different scenarios but also work for robots with different configurations.

Bibliography

[1] Sander Adam, Lucian Busoniu, and Robert Babuska. Experience replay for real-time reinforcement learning control. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 42(2):201–212, 2011.

[2] Alexandre Alahi, Kratarth Goel, Vignesh Ramanathan, Alexandre Robicquet, Li Fei-Fei, and Silvio Savarese. Social lstm: Human trajectory prediction in crowded spaces. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 961–971, 2016.

[3] Javier Alonso-Mora, Andreas Breitenmoser, Paul Beardsley, and Roland Siegwart. Reciprocal collision avoidance for multiple car-like robots. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 360–366. IEEE, 2012.

[4] Haitham Bou Ammar, Eric Eaton, José Marcio Luna, and Paul Ruvolo. Autonomous cross-domain knowledge transfer in lifelong policy gradient reinforcement learning. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015.

[5] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. In *Advances in Neural Information Processing Systems*, pages 5048–5058, 2017.

[6] Rosario Aragues, Jorge Cortes, and C Sagues. Distributed consensus algorithms for merging feature-based maps with limited communication. *Robotics and Autonomous Systems*, 59(3-4):163–180, 2011.

[7] Rosario Aragues, Jorge Cortes, and Carlos Sagues. Distributed consensus on robot networks for dynamically merging feature-based maps. *IEEE Transactions on Robotics*, 28(4):840–854, 2012.

[8] Daman Bareiss and Jur van den Berg. Generalized reciprocal collision avoidance. *The International Journal of Robotics Research*, 34(12):1501–1514, 2015.

[9] Charles Beattie, Joel Z Leibo, Denis Teplyashin, Tom Ward, Marcus Wainwright, Heinrich Küttler, Andrew Lefrancq, Simon Green, Víctor Valdés, Amir Sadik, et al. Deepmind lab. *arXiv preprint arXiv:1612.03801*, 2016.

[10] Nicola Bernini, Massimo Bertozzi, Luca Castangia, Marco Patander, and Mario Sabbatelli. Real-time obstacle detection using stereo vision for autonomous ground vehicles: A survey. In *Intelligent Transportation Systems (ITSC), 2014 IEEE 17th International Conference on*, pages 873–878. IEEE, 2014.

[11] Wolfram Burgard, Mark Moors, Cyrill Stachniss, and Frank E Schneider. Coordinated multi-robot exploration. *IEEE Transactions on robotics*, 21(3):376–386, 2005.

[12] Lucian Buşoniu, Robert Babuška, and Bart De Schutter. Multi-agent reinforcement learning: An overview. In *Innovations in multi-agent systems and applications-1*, pages 183–221. Springer, 2010.

[13] Yongcan Cao, Wenwu Yu, Wei Ren, and Guanrong Chen. An overview of recent progress in the study of distributed multi-agent coordination. *IEEE Transactions on Industrial informatics*, 9(1):427–438, 2013.

[14] Ursula Challita, Walid Saad, and Christian Bettstetter. Cellular-connected uavs over 5g: Deep reinforcement learning for interference management. *arXiv preprint arXiv:1801.05500*, 2018.

[15] Yu Fan Chen, Michael Everett, Miao Liu, and Jonathan P How. Socially aware motion planning with deep reinforcement learning. *arXiv preprint arXiv:1703.08862*, 2017.

[16] Yu Fan Chen, Miao Liu, Michael Everett, and Jonathan P How. Decentralized non-communicating multiagent collision avoidance with deep reinforcement learning. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pages 285–292. IEEE, 2017.

[17] Marc Peter Deisenroth, Peter Englert, Jan Peters, and Dieter Fox. Multi-task policy search for robotics. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3876–3881. IEEE, 2014.

[18] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. Carla: An open urban driving simulator. *arXiv preprint arXiv:1711.03938*, 2017.

[19] Gabriel Dulac-Arnold, Daniel Mankowitz, and Todd Hester. Challenges of real-world reinforcement learning. *arXiv preprint arXiv:1904.12901*, 2019.

[20] Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Volodymir Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, et al. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. *arXiv preprint arXiv:1802.01561*, 2018.

[21] Dave Ferguson, Michael Darms, Chris Urmson, and Sascha Kolski. Detection, prediction, and avoidance of dynamic obstacles in urban environments. In *Intelligent Vehicles Symposium, 2008 IEEE*, pages 1149–1154. IEEE, 2008.

[22] Carlos Florensa, David Held, Markus Wulfmeier, Michael Zhang, and Pieter Abbeel. Reverse curriculum generation for reinforcement learning. *arXiv preprint arXiv:1707.05300*, 2017.

[23] Jakob N Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson. Counterfactual multi-agent policy gradients. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[24] Dieter Fox, Wolfram Burgard, and Sebastian Thrun. The dynamic window approach to collision avoidance. *IEEE Robotics & Automation Magazine*, 4(1):23–33, 1997.

[25] Thierry Fraichard and Alexis Scheuer. Car-like robots and moving obstacles. In *Robotics and Automation, 1994. Proceedings., 1994 IEEE International Conference on*, pages 64–69. IEEE, 1994.

[26] Jayesh K Gupta, Maxim Egorov, and Mykel Kochenderfer. Cooperative multi-agent control using deep reinforcement learning. In *International Conference on Autonomous Agents and Multiagent Systems*, pages 66–83. Springer, 2017.

[27] Matthew Hausknecht and Peter Stone. Deep recurrent q-learning for partially observable mdps. *CoRR, abs/1507.06527*, 7(1), 2015.

[28] Daniel Hennes, Daniel Claes, Wim Meeussen, and Karl Tuyls. Multi-robot collision avoidance with localization uncertainty. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems-Volume 1*, pages 147–154. International Foundation for Autonomous Agents and Multiagent Systems, 2012.

[29] Matteo Hessel, Hubert Soyer, Lasse Espeholt, Wojciech Czarnecki, Simon Schmitt, and Hado van Hasselt. Multi-task deep reinforcement learning with popart. *arXiv preprint arXiv:1809.04474*, 2018.

[30] Todd Hester, Michael Quinlan, and Peter Stone. Rtmba: A real-time model-based reinforcement learning architecture for robot control. In *2012 IEEE International Conference on Robotics and Automation*, pages 85–90. IEEE, 2012.

[31] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[32] Nicholas R Hoff, Amelia Sagoff, Robert J Wood, and Radhika Nagpal. Two foraging algorithms for robot swarms using only local communication. In *Robotics and Biomimetics (ROBIO), 2010 IEEE International Conference on*, pages 123–130. IEEE, 2010.

[33] M Ani Hsieh, Anthony Cowley, Vijay Kumar, and Camillo J Taylor. Maintaining network connectivity and performance in robot teams. *Journal of Field Robotics*, 25(1-2):111–131, 2008.

[34] Franck Iutzeler, Philippe Ciblat, and Jérémie Jakubowicz. Analysis of max-consensus algorithms in wireless channels. *IEEE Transactions on Signal Processing*, 60(11):6103–6107, 2012.

[35] Zilong Jiao and Jae Oh. Simultaneous exploration and harvesting in multi-robot foraging. In *International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems*, pages 496–502. Springer, 2018.

[36] Zilong Jiao and Jae Oh. Asynchronous multitask reinforcement learning with dropout for continuous control. In *2019 18th IEEE International Conference On Machine Learning And Applications (ICMLA)*, pages 529–534. IEEE, 2019.

[37] Zilong Jiao and Jae Oh. End-to-end reinforcement learning for multi-agent continuous control. In *2019 18th IEEE International Conference On Machine Learning And Applications (ICMLA)*, pages 535–540. IEEE, 2019.

[38] Steven Kapturowski, Georg Ostrovski, John Quan, Remi Munos, and Will Dabney. Recurrent experience replay in distributed reinforcement learning. 2018.

[39] Oussama Khatib. Real-time obstacle avoidance for manipulators and mobile robots. In *Autonomous robot vehicles*, pages 396–404. Springer, 1986.

[40] Sujeong Kim, Stephen J Guy, Wenxi Liu, David Wilkie, Rynson WH Lau, Ming C Lin, and Dinesh Manocha. Brvo: Predicting pedestrian trajectories using velocity-space reasoning. *The International Journal of Robotics Research*, 34(2):201–217, 2015.

[41] Nathan Koenig and Andrew Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, volume 3, pages 2149–2154. IEEE, 2004.

[42] Yoram Koren and Johann Borenstein. Potential field methods and their inherent limitations for mobile robot navigation. In *Robotics and Automation, 1991. Proceedings., 1991 IEEE International Conference on*, pages 1398–1404. IEEE, 1991.

[43] Landon Kraemer and Bikramjit Banerjee. Multi-agent reinforcement learning as a rehearsal for decentralized planning. *Neurocomputing*, 190:82–94, 2016.

[44] Hung Manh La, Ronny Lim, and Weihua Sheng. Multirobot cooperative learning for predator avoidance. *IEEE Transactions on Control Systems Technology*, 23(1):52–63, 2015.

[45] Marc Lanctot, Vinicius Zambaldi, Audrunas Gruslys, Angeliki Lazaridou, Karl Tuyls, Julien Pérolat, David Silver, and Thore Graepel. A unified game-theoretic approach to multiagent reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 4190–4203, 2017.

[46] Steven M LaValle. Rapidly-exploring random trees: A new tool for path planning. 1998.

[47] Uichin Lee, Eugenio Magistretti, Mario Gerla, Paolo Bellavista, Pietro Lió, and Kang-Won Lee. Bio-inspired multi-agent data harvesting in a proactive urban monitoring environment. *Ad Hoc Networks*, 7(4):725–741, 2009.

[48] Dan Levi, Noa Garnett, Ethan Fetaya, and Israel Herzlyia. Stixelnet: A deep convolutional network for obstacle detection and road segmentation. In *BMVC*, pages 109–1, 2015.

[49] Lihong Li, Thomas J Walsh, and Michael L Littman. Towards a unified theory of state abstraction for mdps. In *ISAIM*, 2006.

[50] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[51] Pinxin Long, Tingxiang Fan, Xinyi Liao, Wenxi Liu, Hao Zhang, and Jia Pan. Towards optimally decentralized multi-robot collision avoidance via deep reinforcement learning. *arXiv preprint arXiv:1709.10082*, 2017.

[52] Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, OpenAI Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. In *Advances in Neural Information Processing Systems*, pages 6379–6390, 2017.

[53] Tambet Matiisen, Avital Oliver, Taco Cohen, and John Schulman. Teacher-student curriculum learning. *arXiv preprint arXiv:1707.00183*, 2017.

[54] Javier Minguez and Luis Montano. Nearness diagram (nd) navigation: collision avoidance in troublesome scenarios. *IEEE Transactions on Robotics and Automation*, 20(1):45–59, 2004.

[55] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937, 2016.

[56] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[57] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[58] Luc Moreau. Stability of multiagent systems with time-dependent communication links. *IEEE Transactions on automatic control*, 50(2):169–182, 2005.

[59] Johan Nilsson et al. Real-time control systems with delays. 1998.

[60] Reza Olfati-Saber, J Alex Fax, and Richard M Murray. Consensus and cooperation in networked multi-agent systems. *Proceedings of the IEEE*, 95(1):215–233, 2007.

[61] Reza Olfati-Saber and Richard M Murray. Consensus problems in networks of agents with switching topology and time-delays. *IEEE Transactions on automatic control*, 49(9):1520–1533, 2004.

[62] Giovanni Pini, Arne Brutschy, Carlo Pinciroli, Marco Dorigo, and Mauro Birattari. Autonomous task partitioning in robot foraging: an approach based on cost estimation. *Adaptive behavior*, 21(2):118–136, 2013.

[63] Lerrel Pinto and Abhinav Gupta. Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours. In *2016 IEEE international conference on robotics and automation (ICRA)*, pages 3406–3413. IEEE, 2016.

[64] Wang Qian-Ling, Chen Yao, Dong Hai-Rong, Zhou Min, and Ning Bin. A new collision avoidance model for pedestrian dynamics. *Chinese Physics B*, 24(3):038901, 2015.

[65] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y Ng. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, page 5. Kobe, Japan, 2009.

[66] Aravind Rajeswaran, Vikash Kumar, Abhishek Gupta, Giulia Vezzani, John Schulman, Emanuel Todorov, and Sergey Levine. Learning complex dexterous manipulation with deep reinforcement learning and demonstrations. *arXiv preprint arXiv:1709.10087*, 2017.

[67] Tabish Rashid, Mikayel Samvelyan, Christian Schroeder de Witt, Gregory Farquhar, Jakob Foerster, and Shimon Whiteson. Qmix: monotonic value function factorisation for deep multi-agent reinforcement learning. *arXiv preprint arXiv:1803.11485*, 2018.

[68] Benjamin Recht, Christopher Re, Stephen Wright, and Feng Niu. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *Advances in neural information processing systems*, pages 693–701, 2011.

[69] Wei Ren and Randal W Beard. Consensus seeking in multiagent systems under dynamically changing interaction topologies. *IEEE Transactions on automatic control*, 50(5):655–661, 2005.

[70] Steven Adriaan Roelofsen, Denis Gillet, and Alcherio Martinoli. Reciprocal collision avoidance for quadrotors using on-board visual detection. In *International Conference on Intelligent Robots and Systems*, number EPFL-CONF-212951, 2015.

[71] Ahmad EL Sallab, Mohammed Abdou, Etienne Perot, and Senthil Yogamani. Deep reinforcement learning framework for autonomous driving. *Electronic Imaging*, 2017(19):70–76, 2017.

[72] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

[73] Erik Schuitema, Lucian Buşoniu, Robert Babuška, and Pieter Jonker. Control delay in reinforcement learning for real-time dynamic systems: a memoryless approach. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pages 3226–3231. IEEE, 2010.

[74] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015.

[75] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[76] Shital Shah, Debadeepta Dey, Chris Lovett, and Ashish Kapoor. Airsim: High-fidelity visual and physical simulation for autonomous vehicles. In *Field and Service Robotics*, 2017.

[77] Masahiro Shiomi, Francesco Zanlungo, Kotaro Hayashi, and Takayuki Kanda. Towards a socially acceptable collision avoidance for a mobile robot navigating among pedestrians using a pedestrian model. *International Journal of Social Robotics*, 6(3):443–455, 2014.

[78] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484, 2016.

[79] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *ICML*, 2014.

[80] Jamie Snape, Jur Van Den Berg, Stephen J Guy, and Dinesh Manocha. The hybrid reciprocal velocity obstacle. *IEEE Transactions on Robotics*, 27(4):696–706, 2011.

[81] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[82] Peter Sunehag, Guy Lever, Audrunas Gruslys, Wojciech Marian Czarnecki, Vinicius Zambaldi, Max Jaderberg, Marc Lanctot, Nicolas Sonnerat, Joel Z Leibo, Karl Tuyls, et al. Value-decomposition networks for cooperative multi-agent learning. *arXiv preprint arXiv:1706.05296*, 2017.

[83] Lei Tai, Shaohua Li, and Ming Liu. A deep-network solution towards model-less obstacle avoidance. In *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*, pages 2759–2764. IEEE, 2016.

[84] Lei Tai, Giuseppe Paolo, and Ming Liu. Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 31–36. IEEE, 2017.

[85] Ardi Tampuu, Tambet Matiisen, Dorian Kodelja, Ilya Kuzovkin, Kristjan Korjus, Juhan Aru, Jaan Aru, and Raul Vicente. Multiagent cooperation and competition with deep reinforcement learning. *PloS one*, 12(4):e0172395, 2017.

[86] Ming Tan. Multi-agent reinforcement learning: Independent vs. cooperative agents. In *Proceedings of the tenth international conference on machine learning*, pages 330–337, 1993.

[87] Yee Teh, Victor Bapst, Wojciech M Czarnecki, John Quan, James Kirkpatrick, Raia Hadsell, Nicolas Heess, and Razvan Pascanu. Distral: Robust multitask reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 4496–4506, 2017.

[88] Chen Tessler, Shahar Givony, Tom Zahavy, Daniel J Mankowitz, and Shie Mannor. A deep hierarchical approach to lifelong learning in minecraft. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[89] Sebastian Thrun. A probabilistic on-line mapping algorithm for teams of mobile robots. *The International Journal of Robotics Research*, 20(5):335–363, 2001.

[90] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic robotics*. MIT press, 2005.

[91] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 23–30. IEEE, 2017.

[92] Pete Trautman, Jeremy Ma, Richard M Murray, and Andreas Krause. Robot navigation in dense human crowds: Statistical models and experimental studies of human–robot cooperation. *The International Journal of Robotics Research*, 34(3):335–356, 2015.

[93] Karl Tuyls and Gerhard Weiss. Multiagent learning: Basics, challenges, and prospects. *Ai Magazine*, 33(3):41–41, 2012.

[94] Jur Van den Berg, Ming Lin, and Dinesh Manocha. Reciprocal velocity obstacles for real-time multi-agent navigation. In *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, pages 1928–1935. IEEE, 2008.

[95] Jur Van Den Berg, Jamie Snape, Stephen J Guy, and Dinesh Manocha. Reciprocal collision avoidance with acceleration-velocity obstacles. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 3475–3482. IEEE, 2011.

[96] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Thirtieth AAAI conference on artificial intelligence*, 2016.

[97] Hado P van Hasselt, Arthur Guez, Matteo Hessel, Volodymyr Mnih, and David Silver. Learning values across many orders of magnitude. In *Advances in Neural Information Processing Systems*, pages 4287–4295, 2016.

[98] Changyun Wei, Koen V Hindriks, and Catholijn M Jonker. Dynamic task allocation for multi-robot search and retrieval tasks. *Applied Intelligence*, 45(2):383–401, 2016.

[99] Brian Yamauchi. A frontier-based approach for autonomous exploration. In *Computational Intelligence in Robotics and Automation, 1997. CIRA'97., Proceedings., 1997 IEEE International Symposium on*, pages 146–151. IEEE, 1997.

[100] Zhaoyang Yang, Kathryn E Merrick, Hussein A Abbass, and Lianwen Jin. Multi-task deep reinforcement learning for continuous action control. In *IJCAI*, pages 3301–3307, 2017.

[101] Tianhao Zhang, Gregory Kahn, Sergey Levine, and Pieter Abbeel. Learning deep control policies for autonomous aerial vehicles with mpc-guided policy search. In *2016 IEEE international conference on robotics and automation (ICRA)*, pages 528–535. IEEE, 2016.

[102] Yunfei Zhang, Clarence W de Silva, Dijia Su, and Youtai Xue. Autonomous robot navigation with self-learning for collision avoidance with randomly moving obstacles. In *Computer Science & Education (ICCSE), 2014 9th International Conference on*, pages 117–122. IEEE, 2014.

[103] Zhizheng Zhang, Jiale Chen, Zhibo Chen, and Weiping Li. Asynchronous episodic deep deterministic policy gradient: Toward continuous control in computationally complex environments. *IEEE Transactions on Cybernetics*, 2019.

[104] Yuke Zhu, Roozbeh Mottaghi, Eric Kolve, Joseph J Lim, Abhinav Gupta, Li Fei-Fei, and Ali Farhadi. Target-driven visual navigation in indoor scenes using deep reinforcement learning. In *2017 IEEE international conference on robotics and automation (ICRA)*, pages 3357–3364. IEEE, 2017.

VITA

Zilong Jiao was born in Henan, China. He received his Bachelor of Engineering degree at Nanjing University of Posts and Telecommunications (Nanjing, Jiangsu, China) in June 2011. He received his Bachelor of Fine Arts degree at New York Institute of Technology (New York, New York, USA) in May 2011. He received his Master of Science degree in Computer Science at Syracuse University (Syracuse, New York, USA) in May 2014. He Received his Doctor of Philosophy degree in Computer and Information Science and Engineering from Syracuse University (Syracuse, New York, USA) in December 2020.